UNIVERSITY OF CALIFORNIA SAN DIEGO

On the Concrete Security of Identification and Signature Schemes

A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy

in

Computer Science

by

Wei Dai

Committee in charge:

Professor Mihir Bellare, Chair
Professor Nadia Heninger
Professor Russell Impagliazzo
Professor Farinaz Koushanfar
Professor Daniele Micciancio

2021
The dissertation of Wei Dai is approved, and it is acceptable in quality and form for publication on microfilm and electronically.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dissertation Approval Page</td>
<td>iii</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>iv</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vi</td>
</tr>
<tr>
<td>List of Tables</td>
<td>ix</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>x</td>
</tr>
<tr>
<td>Vita</td>
<td>xi</td>
</tr>
<tr>
<td>Abstract of the Dissertation</td>
<td>xii</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Chapter 1: Efficiency Improvements for Big-Key Cryptography</td>
<td>4</td>
</tr>
<tr>
<td>1.1 Introduction</td>
<td>4</td>
</tr>
<tr>
<td>1.2 Preliminaries</td>
<td>11</td>
</tr>
<tr>
<td>1.3 Large-Alphabet Subkey Prediction</td>
<td>13</td>
</tr>
<tr>
<td>1.3.1 The Problem</td>
<td>14</td>
</tr>
<tr>
<td>1.3.2 Subkey Prediction Theorem</td>
<td>15</td>
</tr>
<tr>
<td>1.3.3 Proof of Theorem [1.3.1]</td>
<td>21</td>
</tr>
<tr>
<td>1.3.4 Multi-challenge Subkey Prediction</td>
<td>33</td>
</tr>
<tr>
<td>1.4 Big-Key Symmetric Encryption</td>
<td>35</td>
</tr>
<tr>
<td>1.4.1 Proof of Theorem [1.4.1]</td>
<td>37</td>
</tr>
<tr>
<td>1.5 Big-Key Identification</td>
<td>41</td>
</tr>
<tr>
<td>1.5.1 Proof of Theorem [1.5.1]</td>
<td>52</td>
</tr>
<tr>
<td>1.6 Proofs of Lemmas [1.3.2] [1.3.3] and [1.3.4]</td>
<td>57</td>
</tr>
<tr>
<td>1.7 A Rejection Sampling Lemma</td>
<td>62</td>
</tr>
<tr>
<td>1.8 Acknowledgements</td>
<td>63</td>
</tr>
<tr>
<td>Chapter 2: Tight and Non-rewinding Proofs for Schnorr Identification and Signatures</td>
<td>64</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>64</td>
</tr>
<tr>
<td>2.2 Preliminaries</td>
<td>73</td>
</tr>
<tr>
<td>2.3 The Multi-Base Discrete-Logarithm Problem</td>
<td>75</td>
</tr>
<tr>
<td>2.4 Schnorr Identification and Signatures from MBDL</td>
<td>77</td>
</tr>
<tr>
<td>2.5 MBDL hardness in the Generic Group Model</td>
<td>89</td>
</tr>
<tr>
<td>2.6 Okamoto Identification and Signatures from MBDL</td>
<td>101</td>
</tr>
<tr>
<td>2.7 Ratio-based tightness</td>
<td>104</td>
</tr>
<tr>
<td>2.8 Acknowledgements</td>
<td>106</td>
</tr>
<tr>
<td>Figure 1.1:</td>
<td><strong>Top Left:</strong> Subkey prediction game $G_{skp}^{q,k,\tau}$. <strong>Bottom Left:</strong> Restricted subkey prediction game $G_{rskp}^{k,\tau}$ used in Section 1.3.3. <strong>Right:</strong> Summary of quantities involved.</td>
</tr>
<tr>
<td>Figure 1.2:</td>
<td>Fix the big key length $k^<em>$ to be 100 GBytes. The left graph plots (an upper bound on) $\text{Probes}_{k^</em>,ok^<em>,128(32)}$ as a function of the leakage rate $\rho$. The right graph plots (a lower bound on) $-\log_2(\text{Adv}<em>{skp</em>{2^{32},k,47}(pk)})$ as a function of $\rho$, where $k = k^</em>/32$.</td>
</tr>
<tr>
<td>Figure 1.3:</td>
<td>Multi-challenge subkey prediction game $G_{mcskp}^{q,k,\tau,t}$.</td>
</tr>
<tr>
<td>Figure 1.4:</td>
<td>Game for defining the security of a big-key key encapsulation algorithm $\text{KEY} : {0,1}^k \times {0,1}^r \rightarrow {0,1}^\kappa$.</td>
</tr>
<tr>
<td>Figure 1.5:</td>
<td>Encapsulation algorithm $\text{XKEY}$. Given a length-$k$ big-key $K$ and a length-$r$ selector $R$, the algorithm returns a length-$\kappa$ subkey $K$. The value $\tau$ specifies the number of unique probes used.</td>
</tr>
<tr>
<td>Figure 1.6:</td>
<td>Big-Key Symmetric Encryption Scheme [11, Section 5]. $\text{SE}$, using a standard symmetric key encryption scheme $\text{AE}$ and a key encapsulation mechanism $\text{KEY}$.</td>
</tr>
<tr>
<td>Figure 1.7:</td>
<td>Games $G_m_0, \ldots, G_m_3$. All games share the same procedure $\text{ROR}$ shown on the bottom of the middle column.</td>
</tr>
<tr>
<td>Figure 1.8:</td>
<td>Game $G_m_4$. Note that $T_0$ is a table obtained via coin-fixing.</td>
</tr>
<tr>
<td>Figure 1.9:</td>
<td>Subkey prediction adversary $B$.</td>
</tr>
<tr>
<td>Figure 1.10:</td>
<td>Game defining security of identification scheme $\text{ID}$ under pre-impersonation leakage.</td>
</tr>
<tr>
<td>Figure 1.11:</td>
<td><strong>Left:</strong> Games $G_{cdh}^\text{th}$ and $\text{DL}<em>G$ defining the security of CDH and DL problems in $G$. <strong>Right:</strong> Game $G</em>{pskp}^{p,m,k,\tau}(\mathcal{A},Lk)$. Where $Lk : [q]^k \rightarrow [q]^\ell$ is a leakage function. $[k]^{(\tau)}$ contains the set of $\tau$-dimensional vectors over $[k]$ with distinct entries.</td>
</tr>
<tr>
<td>Figure 1.12:</td>
<td>Algorithms of the ADW identification scheme.</td>
</tr>
<tr>
<td>Figure 1.13:</td>
<td>Game $\text{Rewind}^1$ and $\text{Rewind}^2$ (boxed). The oracle $\text{Leak}$ is the same as the one given in Fig. 1.10.</td>
</tr>
<tr>
<td>Figure 1.14:</td>
<td><strong>Left:</strong> Adversary $\mathcal{A}<em>{cdh}$. <strong>Right:</strong> Adversary $\mathcal{A}</em>{\text{ID}}$.</td>
</tr>
<tr>
<td>Figure 2.1:</td>
<td>Let $\mathcal{G}$ be a group of prime order $p =</td>
</tr>
<tr>
<td>Figure 2.2:</td>
<td><strong>Left:</strong> Algorithm defining an honest execution of the canonical identification scheme $\text{ID}$ given key pair $(sk,pk)$. <strong>Right:</strong> Game defining IMP-PA security of $\text{ID}$.</td>
</tr>
</tbody>
</table>
Figure 2.1: Bounds on ms-uf advantage for 2-round schemes.
Figure 2.2: Chain reductions for multi-signatures.
Figure 2.3: Let $G$ be a group of prime order $p = |G|$ and let $g \in G^*$ be a generator of $G$. The Schnorr ID scheme ID = SchnID[$G,g$] is shown pictorially at the top and algorithmically at the bottom left. At the bottom right is the Schnorr signature scheme DS = SchnSig[$G,g$], using $H : G \times \{0,1\}^* \rightarrow \mathbb{Z}_p$.
Figure 2.4: Top: MBDL adversary $B$ for Theorem 2.4.1 based on IMP-PA adversary $A$.
Figure 2.5: Game defining UF security of signature scheme DS.
Figure 2.6: Game defining $n$-MBDL problem in the generic group model.
Figure 2.7: Games for Lemma 2.5.2.
Figure 2.8: Game $G_{m0}$ for the proof of Theorem 2.5.1. Some procedures will also be in later games, as marked.
Figure 2.9: Procedures for games $G_{m1}, G_{m2}, G_{m3}$ in the proof of Theorem 2.5.1, where $G_{m1}$ includes the boxed code.
Figure 2.10: Procedures for games $G_{m3}, G_{m4}$ in the proof of Theorem 2.5.1. Some procedures, as marked, will be used in later games.
Figure 2.11: Further procedures to define game $G_{m5}$ and games $G_{m\alpha,\beta}$ ($1 \leq \alpha < \beta \leq q$) in the proof of Theorem 2.5.1.
Figure 2.12: Let $G$ be a group of prime order $p = |G|$ and let $g \in G^*$ be a generator of $G$.

The Okamoto ID scheme ID = OkaID[$G,g$] is shown pictorially at the top and algorithmically at the bottom left. At the bottom right is the Okamoto signature scheme DS = OkaSig[$G,g$], using $H : G \times \{0,1\}^* \rightarrow \mathbb{Z}_p$.
Figure 2.13: MBDL adversary $B$ for Theorem 2.6.1 based on IMP-PA adversary $A$.

Figure 3.1: Bounds on ms-uf advantage for 2-round schemes.
Figure 3.2: Chain reductions for multi-signatures.
Figure 3.3: Games defining discrete logarithm, identification logarithm, and random-target identification logarithm problems.
Figure 3.4: **Top left:** Procedure specifying an honest execution of the signing protocol associated with multi-signature scheme MS. **Top right:** Correctness game. **Bottom:** Unforgeability game.
Figure 3.5: Algorithms of the multi-signature scheme BN[$G,g,\ell$] and MuSig[$G,g,\ell$], where $G$ is a group of prime order $p$ with generator $g$. Code that differs between the two schemes is marked explicitly. Oracle $H_i(\cdot)$ is defined to be $H(i,\cdot)$ for $i = 0, 1$ (BN) and $i = 0, 1, 2$ (MuSig).
Figure 3.6: Adversary $A_{ms}$ for Theorem 3.6.1. For an integer $i$, $\langle i \rangle$ denote the binary representation of $i$.
Figure 3.7: Adversary $A_{ms}$ for Theorem 3.6.1. For an integer $i$, $\langle i \rangle$ denote the binary representation of $i$.
Figure 3.8: Two-round multi-signature scheme MS = HBMS[$G,g$] parameterized by a group $G$ of prime order $p$ with generator $g$.
Figure 3.9: Games referred to in Lemma 3.9.1. Both games have just one procedure, Fin, which does not take any input. These games run an algorithm $A$ internally.
Figure 3.10: Games $G_{m0}, G_{m1}, G_{m2}$ and adversary $A_{dl}$ the proof of Theorem 3.3.1.
Figure 3.11: Games $G_{m_0}, G_{m_1}, G_{m_2}$ and adversary $A_{idl}$ for proof of Theorem 3.3.2. $\rho \leftarrow \text{rand}(A_{idl})$ denotes sampling the random coins of $A_{idl}$ and assigning it to $\rho$.

Figure 3.12: Games $G_{m_0}, G_{m_1}, G_{m_2}$ and adversary $A_{idl}$ for proof of Theorem 3.3.3.

Figure 3.13: Games $G_{m_0}, G_{m_1}, G_{m_2}$ and adversary $A_{idl}$ for proof of Theorem 3.3.4.

Figure 3.14: Games $G_{m_0}, G_{m_1}$ for proof of Theorem 3.5.1. Some procedures will be included in later games, as indicated. A box around the name of a game following an oracle means the boxed code in that oracle is included in the game.

Figure 3.15: Games for proof of Theorem 3.5.1.

Figure 3.16: Games for proof of Theorem 3.5.1.

Figure 3.17: Adversary $A_{idl}$ for Theorem 3.5.1.

Figure 3.18: Game $G_{m_{\text{simp}}}$ for proof of Theorem 3.6.1.

Figure 3.19: Games $G_{m_0}, G_{m_1}$ for proof of Theorem 3.6.1. Some procedures will be included in later games, as indicated. A box around the name of a game following an oracle means the boxed code in that oracle is included in the game.

Figure 3.20: Games for proof of Theorem 3.6.1.

Figure 3.21: Games for proof of Theorem 3.6.1.

Figure 3.22: Games for proof of Theorem 3.6.1.

Figure 3.23: Adversary $A_{idl}$ for Theorem 3.6.1. Oracles $\text{NS}, \text{SIGN}_0, \text{SIGN}_1, H_0$ are copied from game $G_{m_{\text{simp}}}$ (Fig. 3.18).

Figure 3.24: Games $G_{m_0}, G_{m_1}, \rho$, and $G_{m_2}, \rho$, where $\rho \in [0, 1]$ is a real number, used in Lemma 3.16.1 and proof of Theorem 3.7.2. Notation $\text{Coin}(\rho)$ denotes flipping of a biased coin with probability $\rho$ of giving 1 and $1 - \rho$ of giving 0.

Figure 3.25: Adversary $A_{idl}$ for Theorem 3.7.1. Oracles $\text{NS}, \text{SIGN}_1, \text{SIGN}_2, H_0, H_1, H_2$ are implemented using the exact code as those in $G_{m_1, \rho}, G_{m_2, \rho}$. Parameter $\rho$ is set to $(1 - (1 + q_s)^{-1})$ in oracle $H_0$.

Figure 3.26: Games $G_{m_3}$ and $G_{m_4}$ for proof of Theorem 3.7.2. Oracles $\text{Init}, \text{NS}, \text{SIGN}_1, \text{SIGN}_2$, and $H_0$ are the same as those in $G_{m_2, \rho}$. Parameter $\rho$ is set to $(1 - (1 + q_s)^{-1})$ in oracle $H_0$.

Figure 3.27: Adversary $A_{idl}$ used in Theorem 3.7.2. Oracles $\text{NS}, \text{SIGN}_1, \text{SIGN}_2, H_0$ are simulated exactly per code from Fig. 3.24.
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1.1</td>
<td>Table comparing efficiencies improvements for different block sizes.</td>
<td>7</td>
</tr>
<tr>
<td>Table 1.2</td>
<td>Table illustrating computation and communication costs and table containing example parameters of the ADW identification scheme.</td>
<td>47</td>
</tr>
<tr>
<td>Table 2.1</td>
<td>Tables comparing speedups yielded by our results for the Schnorr identification and signature schemes.</td>
<td>68</td>
</tr>
<tr>
<td>Table 3.1</td>
<td>Bounds on ms-uf advantage for the 3-round schemes BN and MuSig.</td>
<td>110</td>
</tr>
</tbody>
</table>
I would like to thank all those who have shaped my PhD journey.

First and foremost, I would like to thank my advisor, Mihir Bellare, for his patient guidance and insightful advice, not only regarding technical topics but also regarding problem selection in the broader, societal context.

I would like thank my mentors during my internships that have broaden my academic perspective: Atul Luykx, Stefano Tessaro, Tatsuaki Okamoto, and Go Yamamoto.

I would like to thank all of my co-authors and collaborators (in alphabetical order): Shashank Agrawal, Mihir Bellare, Viet Tung Hoang, Lucy Li, Pratyay Mukherjee, Tatsuaki Okamoto, Peter Rindal, Phillip Rogaway, Stefano Tessaro, Go Yamamoto, and Xihu Zhang.

I would like to thank members of my thesis committee: Mihir Bellare, Nadia Heninger, Russell Impagliazzo, Farinaz Koushanfar, and Daniele Micciancio.

Lastly, I would like to thank students and postdocs of the crypto community at UCSD (in alphabetical order): Vivek Arte, Hannah Davis, Nicholas Genise, Felix Gunther, Joseph Jaeger, Matilda Lundin, Baiyu Li, Ruth Ng, Keegan Ryan, Mark Schultz, Laura Shea, Jessica Sorrell, Igors Stepanovs, Adam Suhl, George Sullivan, Jiahao Sun, Psi Vesely, and Michael Walter.

Chapter 1, in full, is a reprint of the material as it appears in ACM Conference on Computer and Communications Security, 2017. Bellare, Mihir; Dai, Wei. The dissertation author was the primary investigator and author of this paper.

Chapter 2, in full, is a reprint of the material as it appears in International Conference on Cryptology in India, 2020. Bellare, Mihir; Dai, Wei. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in full, is a reprint of the material as it appears in International Conference on the Theory and Application of Cryptology and Information Security, Asiacrypt 2021. Bellare, Mihir; Dai, Wei. Springer, 2021. The dissertation author was the primary investigator and author of this paper.
VITA

<table>
<thead>
<tr>
<th>Year</th>
<th>Degree</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015</td>
<td>Bachelor of Science, in Mathematics and Computer Science, <em>summa cum laude</em>, University of California Santa Barbara</td>
<td></td>
</tr>
<tr>
<td>2016</td>
<td>Master of Science, in Computer Science, University of California Santa Barbara</td>
<td></td>
</tr>
<tr>
<td>2021</td>
<td>Doctor of Philosophy, in Computer Science, University of California San Diego</td>
<td></td>
</tr>
</tbody>
</table>
ABSTRACT OF THE DISSERTATION

On the Concrete Security of Identification and Signature Schemes

by

Wei Dai

Doctor of Philosophy in Computer Science

University of California San Diego, 2021

Professor Mihir Bellare, Chair

Digital signature schemes are ubiquitous in real-world applications of cryptography. They are the core cryptographic building block for public-key infrastructures and distributed ledgers. Yet, the exact security of signature and signature-related schemes are often unknown, due to gaps in their security analyses.

A security proof for a cryptographic scheme S rules out attacks on the scheme assuming hardness of some underlying problem P, for example the discrete-logarithm on elliptic curves. Often, there are gaps between the quantitative security evidenced by cryptanalysis and the quantitative security given by security proofs. For many deployed schemes, quantitative security proofs do not give any meaningful security guarantees. The study of concrete security aims to
eliminate this gap.

In this work, we study the concrete security of (1) a “big-key” identification scheme by Alwen, Dodis, and Wichs, (2) Schnorr identification and signature schemes, and (3) discrete-logarithm-based multi-signature schemes. We identify and tighten the gaps between theoretical guarantees, practical expectations, and best-known cryptanalysis.
Introduction

Digital signature schemes are ubiquitous in real-world applications of cryptography. They are the core cryptographic building block for public-key infrastructures and distributed ledgers. Yet, the exact security of signature and signature-related schemes are often unknown, due to gaps in their security analyses.

A security proof for a cryptographic scheme S rules out attacks on the scheme assuming hardness of some underlying problem P, for example the discrete-logarithm on elliptic curves. Often, there are gaps between the quantitative security evidenced by cryptanalysis and the quantitative security promised by security proofs. For many deployed schemes, quantitative security proofs do not give any meaningful security guarantees. The study of concrete security aims to eliminate this gap.

In this work, we study the concrete security of (1) a “big-key” identification scheme by Alwen, Dodis, and Wichs, (2) Schnorr identification and signature schemes, and (3) discrete-logarithm-based multi-signature schemes. We identify and tighten the gaps between theoretical guarantees, practical expectations, and best-known cryptanalysis.

Efficiency Improvements for Big-Key Cryptography

The first chapter is concerned with the security threat of key exfiltration and the efficiency of schemes achieving the goals of symmetric encryption and public-key identification. Key exfiltration happens when attacker-planted malware on the key-storing system uses the system’s...
network connection to convey the key to a remote accomplice. A line of theoretical work has suggested a mitigation, called the Bounded Retrieval Model (BRM) [41, 38, 30, 6, 5], which involves using big keys. BKR [11] initiated an effort to take the BRM (they call it big-key cryptography) to practicality. We continue this effort.

We first identify probe complexity (the number of scheme accesses to the slow storage medium storing the big key) as the dominant cost for BRM schemes. Our large-alphabet subkey prediction lemma allows us to minimize the probe complexity required to reach a given level of security, thereby optimizing storage usage. We use this to obtain efficiency improvements for big-key symmetric encryption [11]. We then provide an additional lemma on polynomial-evaluation entropy preservation, and use the two lemmas in conjunction to obtain efficiency improvements for the ADW big-key identification scheme [6]. We note that the big-key identification scheme [6] leads to an entropically unforgeable big-key signature signature scheme via the Fiat-Shamir transform, and our efficiency improvements carries over to the signature setting.

Tight and Non-rewinding Proofs for Schnorr Identification and Signature

The second chapter is concerned with the concrete security of Schnorr identification and signature schemes [79]. For these widely-deployed schemes, all known standard-model proofs [76, 1, 58] exhibit a gap: the proven bound on adversary advantage (success probability) is much inferior to (larger than) the one that cryptanalysis says is “true.” (The former is roughly the square-root of the latter. Accordingly we will refer to this as the square-root gap.) The square-root gap is well known and acknowledged in the literature. Filling this long-standing and notorious gap between theory and practice is the subject of this paper.

We introduce the Multi-Base Discrete Logarithm (MBDL) problem. We use this to give reductions, for Schnorr and Okamoto [74] identification and signatures, that are non-rewinding and, by avoiding the notorious square-root loss, tighter than the classical ones from the Discrete Logarithm (DL) problem. This fills a well-known theoretical and practical gap regarding the
security of these schemes. We show that not only is the MBDL problem hard in the generic group model, but with a bound that matches that for DL, so that our new reductions justify the security of these primitives for group sizes in actual use.

**Chain Reductions for Multi-signatures and the HBMS Scheme**

The third chapter is concerned with the concrete security of multi-signature schemes. Usage in cryptocurrencies has lead to interest in practical, Discrete-Log-based multi-signature schemes. Proposals exist, are efficient, and are supported by proofs, but, the bound on adversary advantage in the proofs is so loose that the proofs fail to support use of the schemes in the 256-bit groups in which they are implemented in practice. This leaves the security of in-practice schemes unclear.

We ask, is it possible to bridge this gap to give some valuable support, in the form of tight reductions, for in-practice schemes? As long as we stay in the current paradigm, namely standard-model proofs from DL, the answer is likely NO. To make progress, we need to be willing to change either the model or the assumption. We show that in fact changing either suffices. Our approach is to give, for any scheme, many different paths to security. In particular we give (1) tight reductions from DL in the Algebraic Group Model (AGM) \[47\], and (2) tight, standard-model reductions from well-founded assumptions other than DL. We obtain these results via a framework in which a reduction is “factored” into a chain of sub-reductions involving intermediate problems.

We implement this approach first with classical 3-round schemes, giving chain reductions yielding (1) and (2) above for the BN \[14\] and MuSig \[64\] schemes. Then, in the space of 2-round schemes, we give a new, efficient scheme, called HBMS, for which we do the same.
Chapter 1

Efficiency Improvements for Big-Key Cryptography

1.1 Introduction

This paper is concerned with the threat of key exfiltration. This means attacker-planted malware on the key-storing system uses the system’s network connection to convey the key to a remote accomplice. A line of theoretical work has suggested a mitigation, called the Bounded Retrieval Model (BRM) \([41, 38, 30, 6, 5]\), which involves using big keys. BKR \([11]\) initiated an effort to take the BRM (they call it big-key cryptography) to practicality. We continue this effort. We identify probe complexity (the number of scheme accesses to the slow storage medium storing the big key) as the dominant cost. Our *large-alphabet subkey prediction lemma* allows us to minimize the probe complexity required to reach a given level of security, thereby optimizing storage usage. We use this to obtain efficiency improvements for big-key symmetric encryption \([11]\). We then provide an additional lemma on polynomial-evaluation entropy preservation, and use the two lemmas in conjunction to obtain efficiency improvements for the ADW big-key identification scheme \([6]\).
LARGE-ALPHABET SUBKEY PREDICTION. Let \( b \geq 2 \) be an integer representing the block size in a storage system, for example \( b = 32 \) or \( b = 64 \) for words in memory, or \( b = 8 \cdot 512 \) (512 bytes) for a typical hard-disk drive. Let \( q = 2^b \) be the alphabet size, and \([q] = \{0, 1, \ldots, q - 1\}\) the corresponding alphabet. Let \( \mathbf{K} = (\mathbf{K}[0], \ldots, \mathbf{K}[k-1]) \in \{q\}^k \) be a string over \([q]\) of length \( k \), randomly chosen. It represents a (big) key stored in our storage system as a sequence of \( k \) blocks.

We imagine that an adversary-chosen function \( L_k : \{q\}^k \to \{q\}^\ell \) (representing adversary-implanted malware, and here called the leakage function) is applied to \( \mathbf{K} \), and the result \( L \) (representing exfiltrated information, here called the leakage), is provided back to the adversary. Think of \( \ell \) as somewhat smaller than \( k \), for example \( \ell \leq k/10 \), so that the leakage, although not total, is certainly non-trivial. Despite this, we wish to make secure use of the big key, specifically to (repeatedly) extract “small” keys (\( \tau \geq 1 \) blocks, for a parameter \( \tau \)) for use with conventional cryptography. In any such extraction, we make \( \tau \) random but distinct probes \( i_1, \ldots, i_\tau \in [k] = \{0, 1, \ldots, k-1\} \) into \( \mathbf{K} \) to determine \( J = \mathbf{K}[i_1] \cdots \mathbf{K}[i_\tau] \) as the \( \tau \)-block short key. Given the leakage \( L \) and the probe positions \( i_1, \ldots, i_\tau \), the adversary aims to predict (compute in its entirety) \( J \). Two metrics (see Section 1.3 for precise definitions of what we discuss next) are of interest. First is the subkey prediction advantage

\[
\text{Adv}_{q,k,\tau}^{\text{skp}}(\ell),
\]

defined as the maximum probability that an adversary can compute \( J \), the maximum being over all leakage functions \( L_k \) returning \( \ell \) blocks and over all adversary strategies. It is useful to let \( k^* = kb \) denote the amount of storage occupied by the big key in bits, and, correspondingly, \( \ell^* = \ell b \) the amount of allowed leakage in bits. (We will want to fix these and vary \( b \), thereby defining \( k \) and \( \ell \).) Now, in usage, we would ask for a certain number \( s \) of bits of security (for example \( s = 128 \)), meaning we want the subkey prediction advantage to be at most \( 2^{-s} \), and then want to know the number \( \tau \) of probes it takes to get there. This is the probe complexity,

\[
\text{Probes}_{k^*,\ell^*,s}(b) = \min \left\{ \tau : \text{Adv}_{2^b,k^*,b,\tau}^{\text{skp}}(\ell^*/b) \leq 2^{-s} \right\}.
\]
The probe complexity will be our cost in accesses to a potentially slow storage system, like a
disk, and for efficiency of the overlying big-key scheme, we want to minimize it. To this end,
Theorem 1.3.1 gives a good upper bound on the subkey prediction advantage, whence we obtain
a good upper bound on the probe complexity. Next, we compare our bounds to prior ones, and
discuss history and applications (to big-key cryptography and thus key exfiltration resistance).

**PRIOR WORK AND COMPARISONS.** ADW [7, Lemma A.3] is an elegant and general
result that, as a special case, gives an upper bound on the subkey prediction advantage (and
thus probe complexity) for all values of parameters we consider. The bounds, however are quite
poor, so that, to get a desired level of security, one needs a very large number of probes (we
will see numbers in a bit), resulting in a significant loss of efficiency for the overlying big-key
cryptography schemes. This lead BKR [11] (in their quest for practical big-key symmetric
encryption) to formulate subkey prediction, and seek better bounds by direct analysis. They
however only considered the case $b = 1$ of a binary alphabet. They gave an example to show
that there are non-obvious leakage functions that lead to better subkey prediction advantage
than one might expect, making the problem of giving a (good) upper bound challenging. Via a
combinatorial analysis, they showed that the worst case is when the pre-images of the outputs of
the leakage function are approximate Hamming balls in the space of big keys, thereby deriving
estimates (not quite upper bounds, something we rectify) on the subkey prediction advantage
and probe complexity, for the case $b = 1$ ($q = 2$), that are much better than those obtained via
ADW [6, Lemma A.3]. They posed the large alphabet ($b > 1$) case as an open question, asking,
specifically, to give bounds on subkey prediction advantage and probe complexity, in the $b > 1$
case, that are better than the ones obtained via ADW [7, Lemma A.3]. (The motivation, as we
will see later, was to improve efficiency of big-key symmetric encryption.) Our work answers this
question, giving (good) upper bounds as a function of the block size $b$.

In usage, we would typically first decide on the amount of storage $k^*$ (measured in bits) we
allocate to the big key, for example $k^* = 8 \cdot 10^{11}$ bits = 100 GBytes. Next we would fix the amount
Table 1.1: Fix the amount of storage we allocate to the big key at $k^* = 8 \cdot 10^{11}$ bits = 100 GBytes. Fix the amount of leakage at 10% of the length of the big key, $\ell^* = k^*/10 = 10$ GBytes. The first table considers security level $s = 128$, while the second considers $s = 512$. Each table then considers different block sizes $b$. (Once $b$ is chosen, the length of the big key in blocks is $k = k^*/b$ and the length of the leakage in blocks is $\ell = \ell^*/b$.) The table entries show upper bounds on the probe complexity $\text{Probes}_{k^*,\ell^*,s}(b)$. The “Us” column is our bound via Theorem 1.3.1 and “ADW” is what is obtained via [7, Lemma A.3]. The block sizes are chosen to represent common word or disk sector sizes in storage systems.

<table>
<thead>
<tr>
<th>$b$</th>
<th>$s = 128$</th>
<th>$s = 512$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Us</td>
<td>ADW</td>
</tr>
<tr>
<td>1</td>
<td>271</td>
<td>11532</td>
</tr>
<tr>
<td>8</td>
<td>61</td>
<td>1584</td>
</tr>
<tr>
<td>32</td>
<td>47</td>
<td>592</td>
</tr>
<tr>
<td>64</td>
<td>45</td>
<td>434</td>
</tr>
<tr>
<td>8·512</td>
<td>43</td>
<td>287</td>
</tr>
<tr>
<td>8·4096</td>
<td>43</td>
<td>285</td>
</tr>
</tbody>
</table>

of leakage $\ell^*$ (also measured in bits), for example $\ell^* = k^*/10 = 10$ GBytes, corresponding to 10% of the length of the big key. The block size $b$ may be determined by the storage system (for example 512 bytes or 4096 bytes) or chosen to optimize security and efficiency as per our bounds. Once it is chosen, the length in blocks $k = k^*/b$ of the big key and $\ell = \ell^*/b$ of the leakage are determined. Now, for a given level $s$ of security, we want to know the probe complexity $\text{Probes}_{k^*,\ell^*,s}(b)$. Smaller (fewer probes into the likely slow storage system) is better. We tabulate results in Fig. 1.1. Our bounds emerge as substantially better than those obtained via ADW [7, Lemma A.3]. For example, for $s = 128$, the improvement ranges from a factor of 26 ($b = 8$) to a factor of 6.6 ($b = 8\cdot4096$). Below, we will see how this translates to efficiency improvements for big-key cryptography.

The BRM. Assume (for concreteness of this discussion) that the primitive is symmetric encryption [11] (we will discuss other primitives later), and let $K$ denote the encryption key, $k^*$ bits long. In the Bounded Retrieval Model (BRM) [41, 38, 30, 6, 5, 11], an adversary-chosen function $L_k$ (representing adversary-implanted malware) is applied to $K$, and the $\ell^*$-bit result $L$ (representing the exfiltrated information), is provided back to the adversary. Security would
appear impossible, since \( L_k \) could be the identity function, so that \( L = K \), but the idea is that \( K \) is big (for example \( k^* = 100 \) GBytes), while \( L \) is assumed to be somewhat smaller (like \( \ell^* = k^*/10 = 10 \) GBytes). In other words, the model assumes that the amount of data exfiltrated can be limited, say via network or system monitoring. Indeed, security product vendors such as McAfee [65] provide tools for this type of monitoring and detection.

If the scheme is poorly designed, the fact that the exfiltrated information is somewhat shorter than the key won’t guarantee security. For example if the scheme applies SHA256 to \( K \) to get a 256 bit key \( K \) and then uses AES256 to encrypt the data, then \( L_k(K) \) can just return the 256 bit string \( K = \text{SHA256}(K) \) and security is entirely compromised no matter how big is \( K \). The first requirement for a BRM (also called big-key) scheme is thus leakage resilience, meaning an adversary, given \( L = L_k(K) \), still cannot violate security, and this must be true for any (adversary-chosen) function \( L_k \) that returns \( \ell^* \) bits.

**Probe Complexity.** Big keys may help for security, but it would be prohibitively costly to process a 100 GByte key for every encryption. The BRM addresses this via a condition that says that each encryption (or decryption) operation should only make a “small” number of probes into the big key \( K \), meaning have low probe complexity. Security in the presence of leakage is a difficult goal under any circumstances, but made even more so here by this requirement.

**From Bits to Blocks.** Viewing the big key \( K = (K[0], \ldots, K[k^*-1]) \) as a sequence of bits, BKR encryption [11] begins by making some \( \tau^* \) random probes \( i_1, \ldots, i_{\tau^*} \in [k^*] \) into \( K \) to extract a \( \tau^* \)-bit subkey \( J = K[i_1] \ldots K[i_{\tau^*}] \). It then applies a (randomized) hash function to \( J \) to get a key \( K \) for a conventional (AES-based) symmetric encryption scheme, and uses \( K \) to encrypt the data. Once \( J \) has been obtained, the computation, being symmetric cryptography operations, is quite efficient, but \( K \), being big, is likely stored on a slow medium like a hard drive, and so the encryption cost is dominated by the storage accesses needed to get \( J \). For a subkey prediction advantage of \( s = 128 \) (based on which BKR show ind-cca style security of their encryption scheme at the same security level), BKR will need \( \tau^* = 271 \) probes into the storage.
(This is as per the $b = 1$ row of the first table in Fig. 1.1. BKR’s subkey prediction lemma gives an estimate, not a bound, so we use our number, but numerically the two are almost the same.)

But (as BKR themselves point out), their scheme is making very poor use of storage by drawing only a bit of the big key per probe. Letting $b$ be some appropriate block size determined by the storage system (for example $b = 8 \cdot 512$ bits $= 512$ Bytes), $K$ would actually be stored as a sequence of blocks, and a single probe into the storage can retrieve an entire block at about the same cost as retrieving a single bit. Indeed, a typical storage API does not even provide a way to directly access a bit, so an implementation of BKR would, for a probe for bit-position $j$, have to draw the block containing this bit position, take the corresponding bit, and throw the other bits away. A natural improvement (suggested by BKR) is to draw (and use) an entire block per probe. Thus, we now view the big key $K = (K[0], \ldots, K[k-1]) \in [2^b]^k$ as a sequence of blocks, corresponding to the way it is actually stored, where $k = k^*/b$ is the number of blocks. Now, making some $\tau$ probes $i_1, \ldots, i_\tau \in [k]$ into $K$, one obtains the subkey $J = K[i_1] \ldots K[i_\tau]$. The rest of the encryption process is as before, and as we have already noted, is efficient, even though $J$ could be a bit longer. Continuing to require a subkey prediction advantage of $s = 128$, the question is, what value of $\tau$ guarantees this? This is the question that BKR could not answer. It is answered by our large-alphabet subkey prediction lemma. Specifically, the first table of Fig. 1.1 gives values of $\tau$ for different choices of $b$. For $b = 512$ Bytes, we see that $\tau = 43$. Recalling that BKR needed $\tau^* = 271$ probes, we have reduced the number of probes (storage accesses) by a factor of $271/43 \approx 6$, meaning offer a 6x speedup.

The price we pay (as alluded to above) is that $J$ is longer, specifically, 271 bits for BKR and $43 \cdot 512 \approx 22$ KBytes for us. This means the hashing of $J$ to obtain the AES key $K$ takes longer, but modern hash functions are fast, and the time saved in storage accesses is more than the time lost in extra hashing [31, 34]. This is especially true since the hashing can be pipelined, taking advantage of the iterated structure of hash functions to process blocks incrementally as soon as they are retrieved rather than delaying hashing until after all blocks are retrieved.
**BIG-KEY IDENTIFICATION.** In a (public-key) identification scheme, a user (called the prover) has a secret key $sk$ whose associated public key $pk$ is held by the server (called the verifier). Access control is enforced by having the prover identify itself as the owner of $sk$ via an interactive identification protocol. The Schnorr \[79\] and Okamoto \[74\] schemes are well-known examples, but they are of course conventional (small-key) schemes. Identification is an interesting target for a BRM scheme. Here it is the secret key $sk$ that would be big (100 GBytes)—we want the public key $pk$ to remain of conventional size. The usage we envision is hardware-assisted access control, where $sk$ is on an auxiliary device like a USB stick that the user plugs into a possibly infected machine to identify herself (login) to the server across the network. The key being large, and reading from a USB being slow, the malware will have difficulty obtaining enough information about the key (10 GBytes) to violate BRM security, even after a significant number of usages (logins) by the user.

Identification in the BRM was first treated by ADW \[6\], who gave (asymptotic) security definitions and a clever scheme that involves combining multiple instances of the Okamoto scheme \[74\] in a compact way. We target making this scheme practical. The quest is meaningless in the absence of concrete security, for practicality is fundamentally about maximizing efficiency for a given level (e.g., 128 bits) of security. A first and central step is thus a *concrete-security treatment*. We render the definitions of big-key identification (the goal is security against impersonation under active attack) concretely, then revisit the asymptotically-stated result of ADW \[6\] to render it, too, in concrete form. We note that for the ADW scheme, probe complexity dictates the computational cost of the two most costly phases of the protocol, the response phase and verification phase (as we will demonstrate in Fig. \[1.2\]). Hence, improvements in probe complexity directly translate into improvements in efficiency. Towards lowering probe complexity for a given level of security, we first *improve the concrete security* of the reduction via a lemma on the entropy preservation of polynomial evaluation that improves bounds from ADW \[6\]. We then obtain further reductions in probe complexity, by *using our large-alphabet subkey prediction*.
lemma in place of ADW’s own \cite[Lemma A.3]{7}. The large-alphabet aspect here is crucial, for the scheme draws, from the big key, a value in $\mathbb{Z}_p^m$, where $p$ is a prime of 512 bits long (for 128-bit security of the identification scheme), and $m \geq 2$ is an integer parameter, so probes need to return blocks of the (large) size $b = m \cdot \lceil \log_2(p) \rceil$. Putting it all together gives a reasonable-cost big-key identification scheme, and the first concrete rendition of the ADW big-key identification scheme. A preliminary implementation shows that with a pairing-friendly group of 512 bits, the execution of the protocol takes a few seconds.

1.2 Preliminaries

For $n$ a positive integer, we let $[n] = \{0, 1, \ldots, n - 1\}$, and $[1..n] = \{1, \ldots, n\}$. We also use the notation $\mathbb{Z}_n$ to denote the set $[n]$ in contexts where we use the underlying algebraic structure modulo $n$. If $x$ is a vector, then $|x|$ denotes its length and $x[i]$ denotes its $i$-th coordinate. We call $x$ an $n$-vector if $|x| = n$. We number coordinates starting from 0. For example if $x = (10, 0, 11)$ then $|x| = 3$ and $x[2] = 11$. We let $\varepsilon$ denote the empty vector, which has length 0. If $0 \leq i \leq |x| - 1$ then we let $x[0..i] = (x[0], \ldots, x[i])$, this being $\varepsilon$ when $i = 0$. We say that $x$ is a vector over set $S$ if all its coordinates belong to $S$. We let $S^n$ denote the set of all $n$-vectors over $S$ and we let $S^*$ denote the set of all finite-length vectors over the set $S$. If $S$ is a set then $|S|$ denotes its size. If $\tau \leq |S|$ is a positive integer, we let $S^{(\tau)}$ be the set of $\tau$-vectors over $S$ with distinct entries. Strings are treated as the special case of vectors over $\{0, 1\}$. Thus, if $x$ is a string then $|x|$ is its length, $x[i]$ is its $i$-th bit, $x[0..i] = x[0]...x[i]$, $\varepsilon$ is the empty string, $\{0, 1\}^n$ is the set of $n$-bit strings and $\{0, 1\}^*$ the set of all strings. For $K \in [q]^k$ and $pp \in [k]^*$, we let $K[pp] = (K[pp[0]], K[pp[1]], \ldots, K[pp[|pp| - 1]])$, this being $\varepsilon$ when $pp = \varepsilon$.

If $X$ is a finite set, we let $x \leftarrow X$ denote picking an element of $X$ uniformly at random and assigning it to $x$. Algorithms may be randomized unless otherwise indicated. Running time is worst case. If $A$ is an algorithm, we let $y \leftarrow A(x_1, \cdots ; r)$ denote running $A$ with random coins $r$
on inputs $x_1, \cdots$ and assigning the output to $y$. We let $y \leftarrow A(x_1, \cdots)$ be the result of picking $r$ at random and letting $y \leftarrow A(x_1, \cdots; r)$. We let $[A(x_1, \cdots)]$ denote the set of all possible outputs of $A$ when invoked with inputs $x_1, \cdots$.

We use the code-based game-playing framework [19] (see Fig. 1.1 for an example). By $\Pr[\text{Gm}]$ we denote the probability that game Gm returns true. Uninitialized boolean variables, sets and integers are assume initialized to false, the empty set and 0, respectively.

Following [17], our random oracle $H$ is variable range. This means it takes two inputs, $x$ and $\text{Img}$, where the latter is a (description of) an efficiently sampleable set, and returns as output a random point in $\text{Img}$. In schemes, we (implicitly or explicitly) fix a unique description for each range set that is used. For example, $x \leftarrow H(x, [k])$ will return a random element in $[k]$, and $pp \leftarrow H(x, [k]^{(\tau)})$ will return a random $\tau$-dimensional vector over $[k]$ with distinct entries.

**Hamming Balls.** Let $q \geq 2$ and $n \geq 1$ be integers. We define the weight of a $n$-vector $v$ over $[q]$ to be

$$w(v) = \left| \{i \in [n] \mid v[i] \neq 0 \} \right|,$$

the number of coordinates of $v$ that are non-zero. Let $\mathcal{K} \subseteq [q]^n$ for some integer $n$, we define the weight of $\mathcal{K}$ to be

$$w(\mathcal{K}) = \sum_{x \in \mathcal{K}} w(x),$$

the sum of weights of vectors in $\mathcal{K}$. For $0 \leq r \leq k$, the $q$-ary hamming ball of radius $r$ over $[q]^k$ is the set

$$B_{q,k}(r) = \left\{ v \in [q]^k : w(v) \leq r \right\}$$

of $k$-vectors over $[q]$ that have more at most $r$ non-zero coordinates. We let $B_{q,k}(r)$ denote the size of the set $B_{q,k}(r)$ and note that

$$B_{q,k}(r) = \sum_{i=0}^{r} (q - 1)^i \binom{k}{i}.$$
The alphabet size. A block is an element of $[q]$. $k$ : the length in blocks of the big key $\tau \leq k$ : the number probes into the big key $\mathbf{K}$ $A$ : the adversary $Lk : [q]^k \rightarrow [q]^\ell$ : the leakage function $L : [q]^\ell \rightarrow [q]^\ell$ : the leakage, an \( \ell \)-vector over $[q]$ returned by $Lk$ $\mathbf{K}$ : the big key, a vector of length $k$ over $[q]$ $p$ : the probe vector, a $\tau$-vector over $[k]$ all of whose coordinates are distinct $k^* : the length of the big key in bits, $k^* = kb$ $\ell^* : the length of the leakage in bits, $\ell^* = \ell b$ $\rho : the leakage rate, $\rho = \ell^* / k^* = \ell / k$

| Game $G_{q,k,\tau}^{skp}(A,Lk)$ | $q \geq 2$ : the alphabet size. A block is an element of $[q]$. $k$ : the length in blocks of the big key $\tau \leq k$ : the number probes into the big key $\mathbf{K}$ $A$ : the adversary $Lk : [q]^k \rightarrow [q]^\ell$ : the leakage function, $Lk : [q]^k \rightarrow [q]^\ell$ $\ell$ : the length of the output of the leakage function, called the leakage length, in blocks $b \geq 1$ : the block length, meaning $q = 2^b$. Theorem 1.3.1 does not assume $q$ is a power of two, but it is in some applications. $L : the leakage, an \ell$-vector over $[q]$ returned by $Lk$ $\mathbf{K}$ : the big key, a vector of length $k$ over $[q]$ $p$ : the probe vector, a $\tau$-vector over $[k]$ all of whose coordinates are distinct $k^* : the length of the big key in bits, $k^* = kb$ $\ell^* : the length of the leakage in bits, $\ell^* = \ell b$ $\rho : the leakage rate, $\rho = \ell^* / k^* = \ell / k$

| Game $G_{k,\tau}^{rskp}(A, K)$ | $\mathbf{K}$ : the big key, a vector of length $k$ over $[q]$ $p$ : the probe vector, a $\tau$-vector over $[k]$ all of whose coordinates are distinct $k^* : the length of the big key in bits, $k^* = kb$ $\ell^* : the length of the leakage in bits, $\ell^* = \ell b$ $\rho : the leakage rate, $\rho = \ell^* / k^* = \ell / k$

**Figure 1.1**: Top Left: Subkey prediction game $G_{q,k,\tau}^{skp}$. Bottom Left: Restricted subkey prediction game $G_{k,\tau}^{rskp}$ used in Section 1.3.3. Right: Summary of quantities involved.

For convenience of stating our results, we establish the following conventions: if $r > k$ then we let $B_{q,k}(r) = B_{q,k}(k) = q^k$, and if $k = 0$ then for all $r \geq 0$ we let $B_{q,k}(r) = 1$. We also define the function

$$rd_{q,k}(N) = \max \{ r \in [k+1] : B_{q,k}(r) \leq N \}$$

to return the largest radius $r$ in the range $0 \leq r \leq k$ such that the ball $B_{q,k}(r)$ has size at most $N$.

### 1.3 Large-Alphabet Subkey Prediction

Here we define the subkey prediction problem parameterized by alphabet size and give our results about it.
1.3.1 The Problem

We consider the subkey-prediction game, $G_{skp q, k, \tau}^A (\mathcal{L}_k)$, shown on the top left of Fig. 1.1 (Ignore the game below it for now.) The quantities involved in the game, as well as associated ones, are summarized on the right of the same Figure. In the game, a $k$-vector $K$ over $[q]$, called the big key, is randomly chosen from $[q]_k$. Then, a random $\tau$-vector $p$ is chosen from $[k]^{(\tau)}$, so that its coordinates are all distinct. (Recall that $[k]^{(\tau)}$, the set from which $p$ is selected in the game in Fig. 1.1, denotes the set of all $\tau$-vectors over $[k]$ all of whose coordinates are distinct.) We refer to $p$ as the probe vector. Each of its coordinates is a probe, pointing to a location in the big key. Adversary $A$ is given the leakage $L = \mathcal{L}_k(K)$ and the probe vector $p$. Its goal is to predict (compute, output) $K[p] = (K[p[1]], \ldots, K[p[\tau]])$, the $\tau$-vector consisting of the coordinates of $K$ selected by the coordinates of the probe vector. The adversary returns $J$ as its guess, and the game returns true if $A$ succeeds, meaning $J = K[p]$. We define the following advantage metrics:

$$
\text{Adv}_{skp q, k, \tau}^A (\mathcal{L}_k) = \Pr \left[ G_{skp q, k, \tau}^A (\mathcal{L}_k) \right],
$$

$$
\text{Adv}_{skp q, k, \tau}^\mathcal{L}_k = \max_{\mathcal{A}} \text{Adv}_{skp q, k, \tau}^A (\mathcal{L}_k),
$$

$$
\text{Adv}_{skp q, k, \tau}^\ell = \max_{\mathcal{L}_k: [q]^k \to [q]^\ell} \text{Adv}_{skp q, k, \tau}^\mathcal{L}_k.
$$

The first advantage is the probability that the game outputs true, meaning the probability that the adversary successfully returns $K[p]$. The second advantage is obtained by maximizing the first one over all adversaries $A$. Note that this is well-defined since here we consider all computationally unbounded adversaries. The third advantage is obtained by maximizing the second advantage over all leakage functions $\mathcal{L}_k$ that output $\ell$ blocks.

Now fix some big-key length $k^*$ (in bits) and leakage length $\ell^*$ (in bits). Also fix an integer $s$ representing the desired security. For any block length $b \geq 1$ such that $b$ divides $k^*$ and $\ell^*$, we
Probes}_{k^*,\ell^*,s}(b) = \min \left\{ \tau : \text{Adv}_{2^k,k^*/b,\tau}(\ell^*/b) \leq 2^{-s} \right\}.

(1.3)

Here, we have set the alphabet size to $q = 2^b$. The length $k$ of the big key and $\ell$ of the leakage in blocks are determined, respectively, by $k = k^*/b$ and $\ell = \ell^*/b$. Then, Probes}_{k^*,\ell^*,s}(b)$ is the smallest number of probes $\tau$ that will guarantee that $\text{Adv}_{q,k,\tau}(\ell)$ is at most $2^{-s}$.

The subkey prediction game and problem formulated by BKR [11] differs in two ways. First, they had only considered the $q = 2$ case (that is, $b = 1$) of a binary alphabet. The large alphabet aspect of our treatment refers to the fact that our alphabet size is a parameter $q$ that we view as quite large. In some applications, $q = 2^b$ where $b$ is the block size of our storage medium, but Theorem [1.3.1] does not assume $q$ is a power of two. The second difference with BKR [11] is that their probes $p[1], \ldots, p[\tau]$ were random and independent, so in particular two of them might be the same, but ours are random subject to being distinct. This is important towards our being able to get a provable upper bound on the subkey prediction advantage, whereas BKR were only able to get (for their setting) an estimate or approximate upper bound.

Now our goal is to upper bound, as well as possible, the subkey prediction advantage $\text{Adv}_{q,k,\tau}(\ell)$ as a function of $q, k, \tau, \ell$. Thence we will obtain upper bounds on $\text{Probes}_{k^*,\ell^*,s}(b)$.

1.3.2 Subkey Prediction Theorem

The bound in our subkey prediction theorem is the ratio of the sizes of two $q$-ary hamming balls. We refer to Section [1.2] for definitions.

**Theorem 1.3.1 (Subkey-prediction bound)** Let $q, k, \ell, \tau$ be integers with $q \geq 2$ and $\ell, \tau \leq k$. Let $r$ be any integer in the range $0 \leq r \leq rd_{q,k}(q^{k-\ell})$. Then

$$\text{Adv}_{q,k,\tau}(\ell) \leq \frac{B_{q,k-\tau}(r)}{B_{q,k}(r)}.$$ (1.4)
The theorem allows us to pick the parameter \( r \) arbitrarily in the given range, so for the best estimates we would pick a \( r \) that minimizes the ratio. We postpone the proof to first discuss how this compares to prior work and how to use it to get numerical bounds.

**Comparison.** BKR [11] give an upper bound we denote \( G_{k,\tau}^{bkr}(2^{k-\ell}) \) on the subkey prediction advantage in their setting. Recall that their setting differs from ours in two ways. First, \( q = 2 \) in their case. Second, in their game, the \( \tau \) probes are random and independent, while in our game they are random but distinct. Their function \( G_{k,\tau}^{bkr}(N) \) is a sum of \( r d_{2,k}(N) \) terms. It is quite complex and it is hard to estimate numerically. BKR gave a simpler expression, that approximates \( G_{k,\tau}^{bkr}(N) \), and that they use for numerical estimates, but this expression is not an upper bound, and thus it is not clear their numerical estimates are upper bounds either. Our bound, the ratio of the sizes of two \( q \)-ary Hamming balls, is simpler than the bound of BKR (this makes crucial use of the probes being distinct), and, we will see, more analytically tractable, even when \( q = 2 \). In particular, we are able to upper bound \( \min_r B_{q,k-\tau}(r)/B_{q,k}(r) \), subjected to \( 0 \leq r \leq r d_{q,k}(q^{k-\ell}) \), quite nicely for numerical estimates, as discussed next.

**Tools for Deriving Numerical Bounds.** Theorem [1.3.1] upper bounds the subkey prediction advantage as the ratio of the sizes of two hamming balls. Below, we present tools to bound this ratio. First, we need some definitions. Let \( H_2 \) be the binary entropy function, defined for \( x \in [0,1] \) by

\[
H_2(x) = -x \log_2(x) - (1-x) \log_2(1-x).
\]

We note that the value of \( x \log_2(x) \) is taken to be 0 when \( x = 0 \). This ensures that \( H_2 \) is continuous over \([0,1]\). More generally, for an integer \( q \geq 2 \) the \( q \)-ary entropy function is defined for \( x \in [0,1] \) by

\[
H_q(x) = x \log_q(q-1) - x \log_q(x) - (1-x) \log_q(1-x)
= \frac{H_2(x)}{\log_2(q)} + x \log_q(q-1).
\]

We note that \( H_q \) attains its maximum at \( x = 1 - 1/q \). We define its inverse function, \( H_q^{-1} : [0,1] \to [0,1-1/q] \) to be such that \( H_q^{-1}(H_q(x)) = x \) for any \( x \in [0,1 - 1/q] \). We define the following...
error function for $q \geq 2$ and $0 \leq r \leq k$,

$$
\varepsilon(q, k, r) = \log_q(e) \left( \frac{1}{12r} + \frac{1}{12(k-r)} - \frac{1}{12k+1} \right) + \frac{1}{2} \log_q \left( \frac{2\pi r(k-r)}{k} \right). \tag{1.5}
$$

The following lemmas, the proofs of which are given in Section 1.6, are key to deriving numerical bounds. The first gives both upper and lower bounds on the size of a Hamming ball.

**Lemma 1.3.2** Let $k, q, r$ be integers with $q \geq 2$ and $0 \leq r \leq k$. Then,

$$
q^{H_q(r/k) - \varepsilon(q,k,r)} \leq B_{q,k}(r). \tag{1.6}
$$

Additionally, if $0 \leq r \leq k(1 - 1/q)$,

$$
B_{q,k}(r) \leq q^{H_q(r/k)}. \tag{1.7}
$$

The second lemma lower bounds the value of $r d_{q,k}(N)$.

**Lemma 1.3.3** Let $N, q, k$ be positive integers such that $q \geq 2$ and $N \leq q^k$. Then,

$$
\left\lfloor H_q^{-1} \left( \frac{\log_q(N)}{k} \right) \cdot k \right\rfloor \leq r d_{q,k}(N).
$$

The following provides a two-sided bound on $H_q^{-1}$:

**Lemma 1.3.4** Let $q \geq 2$ be an integer, and $x \in [0, 1]$ a real number. Then,

$$
\min(x, 1 - \frac{1}{q}) - \frac{1}{\log_2(q)} \leq H_q^{-1}(x) \leq x(1 - \frac{1}{q}).
$$

These bounds are good when $q$ is large.

**DERIVING NUMERICAL BOUNDS.** We now use the above to derive upper bounds for example parameter values. Let $b \geq 1$ be a block size, so that the alphabet has size $q = 2^b$. Fix
some big-key length $k^*$ (in bits) and leakage length $\ell^*$ (in bits) that are multiples of $b$, and let $k = k^*/b$ and $\ell = \ell^*/b$ be the big-key and leakage lengths, respectively, in blocks. We assume that $\tau$ and $\ell$ satisfy that $\ell \geq \tau$, as the below method only apply when this condition is met. We note that this is a reasonable assumption for practical applications, as leakage length $\ell$ is usually large, and we are attempting to keep the probe complexity, $\tau$, small. Now, suppose we have obtained some integer value $r$ such that: (1) $r \leq rd_{q,k}(q^{k-\ell})$ and (2) $0 \leq r \leq (k-\tau)(1-1/q)$. Then, we use Equation [1.6] to lower bound $B_{q,k}(r)$. Given condition (2), we can use Equation [1.7] to upper bound the quantity $B_{q,k-\tau}(r)$. This results in an upper bound, denoted $\text{RatioBound}_{q,k,\ell,\tau}(r)$, for the ratio $B_{q,k-\tau}(r)/B_{q,k}(r)$:

$$\text{RatioBound}_{q,k,\ell,\tau}(r) = \frac{q^{k-\tau}H_{q}(r/(k-\tau))}{q^{kH_{q}(r/k)-\varepsilon(q,k,r)}}.$$ 

Note that in the above expression, the terms $H_{q}(r/(k-\tau))$, $H_{q}(r/k)$ and $\varepsilon(q,k,r)$ can be computed numerically for any given value of $q,k,\tau$ and $r$. Hence, deriving numerical upper bound for the ratio $B_{q,k-\tau}(r)/B_{q,k}(r)$ amounts to obtaining a value $r$ satisfying the two conditions given above. We take $r$ to be $r_{q,k,\ell}$, defined as

$$r_{q,k,\ell} = \left[H_{q}^{-1}\left(\frac{k-\ell}{k}\right) \cdot k\right].$$

Here, we assume that a method of obtaining numerical lower bounds for $H_{q}^{-1}(x)$ is available. We now check the two conditions required. For condition (1), we know that $r_{q,k,\ell} \leq rd_{q,k}(q^{k-\ell})$ by Lemma [1.3.3] (taking $N = q^{k-\ell}$). For condition (2), note that by Lemma [1.3.4] and the assumption that $\ell \geq \tau$,

$$H_{q}^{-1}\left(\frac{k-\ell}{k}\right) \leq \frac{k-\ell}{k} (1-1/q) \leq \frac{k-\tau}{k} (1-1/q).$$

\[1\] For example, this is available in mathematical software Sage. Also, when $q$ is large, Lemma [1.3.4] provides a good lower bound for $H_{q}^{-1}$ that is easily computed numerically.
Hence,

\[ r_{q,k,\ell} = \left\lfloor H^{-1}_q\left(\frac{k-\ell}{k}\right) \cdot k \right\rfloor \leq (k-\tau)(1-1/q). \]

We consider the quantity

\[ \text{Adv}^{\text{skp}}_{q,k,\tau}(\ell) = \text{RatioBound}_{q,k,\tau}(r_{q,k,\ell}). \tag{1.8} \]

We note that since \( r = r_{q,k,\ell} \) satisfies condition (1) and (2), by Theorem 1.3.1 and above analysis,

\[ \text{Adv}^{\text{skp}}_{q,k,\tau}(\ell) \leq \frac{B_{q,k-\tau}(r_{q,k,\ell})}{B_{q,k}(r_{q,k,\ell})} \leq \text{Adv}^{\text{skp}}_{q,k,\tau}(\ell). \]

Hence, \( \text{Adv}^{\text{skp}}_{q,k,\tau}(\ell) \) is an upper bound for \( \text{Adv}^{\text{skp}}_{q,k,\tau}(\ell) \). Now, given a particular desired security level, \( s \), we want to find the smallest \( \tau \) such that \( \text{Adv}^{\text{skp}}_{q,k,\tau}(\ell) \leq 2^{-s} \). We let

\[ \text{Probes}_{k^*,\ell^*,b}^{s}(s) = \min\{\tau \in [k+1] : \text{Adv}^{\text{skp}}_{q,k,\tau}(\ell) \leq 2^{-s}\}. \]

Note that this is similar to the definition of \( \text{Probes}_{k^*,\ell^*,b}^{s}(s) \) (Equation 1.3), only that \( \text{Adv}^{\text{skp}}_{q,k,\tau}(\ell) \) is replaced with \( \text{Adv}^{\text{skp}}_{q,k,\tau}(\ell) \). Thence,

\[ \text{Probes}_{k^*,\ell^*,b}^{s}(s) \leq \text{Probes}_{k^*,\ell^*,b}^{s}(s). \tag{1.9} \]

We note that \( \text{Probes}_{k^*,\ell^*,b}^{s}(s) \) can be computed numerically by iteratively incrementing \( \tau \) and computing \( \text{Adv}^{\text{skp}}_{q,k,\tau}(\ell) \). Fig. 1.1 shows values of \( \text{Probes}_{k^*,\ell^*,b}^{s}(s) \) for various practical values of \( k^*, \ell^*, b \) and \( s \).

**Plots.** For the left plot, we fix the following:

- Blocksize \( b = 32 \) bits, so that \( q = 2^{32} \).
- Leakage length \( \ell^* = 8 \cdot 10^{10} \) bits = 10 GBytes, so that \( \ell = \ell^*/32 \).
Figure 1.2: Fix the big key length $k^*$ to be 100 GBytes. The left graph plots (an upper bound on) $\text{Probes}_{k^*, \rho^*, 128(32)}$ as a function of the leakage rate $\rho$. The right graph plots (a lower bound on) $-\log_2(\text{Adv}_{skp, q,k,47}^{\text{skp}}(\ell))$ as a function of $\rho$, where $k = k^*/32$.

- Desired security level $s = 128$ bits.

The left graph in Fig. 1.2 plots $\text{Probes}_{\ell^*/\rho, \ell^*, b}(s)$, upper bound for $\text{Probes}_{\ell^*/\rho, \ell^*, b}(s)$, as a function of the leakage rate $\rho$. The left plot shows that the number of probes needed to maintain $s$ bits of security increases faster once the leakage rate goes over 50%. Hence, for applications, it may be beneficial to use big keys that are big enough so that the leakage rate can be assumed to be less than 50%. For example, if 10 GBytes is the leakage bound, one might, for efficiency, target big key of size at least 20 GBytes.

For the right plot, we fix the following
- Blocksize $b = 32$ bits, so that $q = 2^{32}$.
- Big key length $k^* = 8 \cdot 10^{11}$ bits = 100 GBytes, so that $k = k^*/32$.
- Number of probes $\tau = 47$.

The number 47 has been chosen because, as per Fig. 1.1, it ensures $\text{Adv}_{q,k,\tau}(k/10) \leq 2^{-128}$. Now with $b, k^*, \tau$ (and thus also $q, k$) fixed, the right graph plots $-\log_2(\text{Adv}_{skp, q,k,\tau}(\rho \cdot k))$, lower bound for $-\log_2(\text{Adv}_{skp, q,k,\tau}(\rho \cdot k))$, as a function of leakage rate $\rho$. The right plot in Fig. 1.2 demonstrates that, even though a scheme is designed for 10% leakage, security degrades gradually as the leakage rate goes over 10%.
1.3.3 Proof of Theorem 1.3.1

We follow the framework of the proof of BKR [11].

**RESTRICTED SUBKEY PREDICTION.** The proof involves consideration of a simpler game, called the restricted subkey prediction game, denoted \( G^{rskp} \) and shown on the right in Fig. 1.1. Game \( G^{rskp} \) is similar to game \( G^{skp} \), except that there is no leakage function \( L_k \) and leakage \( L \). Instead, the big key \( K \) is drawn from a restricted subset \( K \subseteq [q]^k \) of big keys. We define the following advantage metrics:

\[
\text{Adv}_{k, \tau}^{rskp}(A, \mathcal{K}) = \Pr \left[ G_{k, \tau}^{rskp}(A, \mathcal{K}) \right],
\]

\[
\text{Adv}_{k, \tau}^{rskp}(\mathcal{K}) = \max_A \text{Adv}_{k, \tau}^{rskp}(A, \mathcal{K}),
\]

\[
\text{Adv}_{q, k, \tau}(N) = \max_{\mathcal{K} \subseteq [q]^k, |\mathcal{K}| = N} \text{Adv}_{k, \tau}^{rskp}(\mathcal{K}).
\]

The first advantage is the probability that the game outputs true, meaning the probability that the adversary successfully returns \( K[p] \). The second advantage is obtained by maximizing the first one over all adversaries \( A \). The third advantage is obtained by maximizing the second advantage over all sets \( \mathcal{K} \subseteq [q]^k \) that have size \( N \). We note that the first two advantages do not have \( q \) in the subscript, which is due to the fact that \( \mathcal{K} \) encodes the value of \( q \).

**MONOTONE SETS.** Let \( x, x' \) be vectors in \( [q]^k \). We say that \( x \) dominates \( x' \), or \( x' \) is dominated by \( x \), written \( x' \leq x \), if \( x' \) can be obtained by changing non-zero coordinates of \( x \) to 0. We let

\[
\text{DS}_{q, k}(x) = \{ x' \in [q]^k : x' \leq x \}
\]

be the set of all \( x' \) dominated by \( x \). A set \( \mathcal{K} \subseteq [q]^k \) is **monotone** if

\[
\bigcup_{x \in \mathcal{K}} \text{DS}_{q, k}(x) \subseteq \mathcal{K}.
\]

That is, if \( x \in \mathcal{K} \), and \( x' \) is dominated by \( x \), then \( x' \in \mathcal{K} \). For example, a Hamming ball in \( [q]^k \), of
any radius, is a monotone set.

** SOME NOTATION.** For integers \( x, \tau \geq 0 \), we let

\[
x_{(\tau)} = \prod_{i=0}^{\tau-1}(x - i) = \prod_{j=x-\tau+1}^{x} j. \tag{1.10}
\]

Notice that \( x_{(\tau)} = 0 \) if \( \tau > x \). This can be seen because, if \( \tau > x \), then, in the second product above, the starting value for \( j \) is \( \leq 0 \), and since \( x \geq 0 \), this means the term \( j = 0 \) is included in the product. Also when \( \tau = 0 \), the product has zero terms, and hence by convention takes value 1, meaning \( x_{(0)} = 1 \) for all \( x \geq 0 \). We use below the notation from Equation (1.10).

For a nonempty \( \mathcal{K} \subseteq [q]^k \), we define the function

\[
g_{k,\tau}(\mathcal{K}) = \frac{1}{|\mathcal{K}|} \sum_{x \in \mathcal{K}} \frac{(k - w(x))_{(\tau)}}{k_{(\tau)}}. \tag{1.11}
\]

The following lemma says that if \( \mathcal{K} \) is monotone, then the restricted subkey prediction advantage for big keys drawn from \( \mathcal{K} \) can be expressed exactly, and in particular by the function of Equation (1.11).

**Lemma 1.3.5** Let \( q, \tau, k \) be positive integers such that \( \tau \leq k \) and \( q \geq 2 \). Let \( \mathcal{K} \subseteq [q]^k \) be a non-empty monotone set. Then,

\[
\text{Adv}^{rskp}_{k,\tau}(\mathcal{K}) = g_{k,\tau}(\mathcal{K}).
\]

**Proof of Lemma 1.3.5** Let \( \mathcal{A}_0 \) be the adversary that, on input \( p \), always returns the all-0 \( \tau \)-vector. We claim that this adversary maximizes the advantage, meaning

\[
\text{Adv}^{rskp}_{k,\tau}(\mathcal{K}) = \text{Adv}^{rskp}_{k,\tau}(\mathcal{K}, \mathcal{A}_0).
\]

This follows from the assumption that \( \mathcal{K} \) is monotone. Now, we compute the advantage of \( \mathcal{A}_0 \).
For $K \in [q]^k$, let $Z(K)$ denote the set of all $p \in [k]^{(\tau)}$ such that $K[pp] = (0, \ldots, 0)$. We have

$$\text{Adv}_{rskp}^r(K, \mathcal{A}_0) = \frac{1}{|\mathcal{K}|} \sum_{K \in \mathcal{K}} |Z(K)| = \frac{1}{|\mathcal{K}|} \sum_{K \in \mathcal{K}} \frac{(k - w(K))_{(\tau)}}{k_{(\tau)}} = g_{k, \tau}(\mathcal{K}).$$

We say that a set $\mathcal{K} \subseteq [q]^k$ is sandwiched between hamming balls if

$$B_{q,k}(r) \subseteq \mathcal{K} \subseteq B_{q,k}(r + 1)$$

for $r = rd_{q,k}(|\mathcal{K}|)$. For $N$ an integer such that $1 \leq N \leq q^k$, we define

$$G_{q,\tau,k}(N) = \frac{1}{N} \sum_{i=0}^{rd_{q,k}(N)} (q-1)^i \binom{k}{i} \frac{(k-i)_{(\tau)}}{k_{(\tau)}} + \left(1 - \frac{B_{q,k}(rd_{q,k}(N))}{N}\right) \frac{(k-(rd_{q,k}(N)+1))_{(\tau)}}{k_{(\tau)}} .$$

The following says that if $\mathcal{K}$ is monotone and sandwiched between Hamming balls, then the restricted subkey prediction advantage for big keys drawn from $\mathcal{K}$ can be expressed exactly, and in particular by the function of Equation 1.12.

**Lemma 1.3.6** Let $q, \tau, k$ be positive integers such that $\tau \leq k$ and $q \geq 2$. Let $\mathcal{K} \subseteq [q]^k$ be a non-empty monotone set that is also sandwiched between hamming balls, i.e. $B_{q,k}(r) \subseteq \mathcal{K} \subset B_{q,k}(r + 1)$ for $r = rd_{q,k}(|\mathcal{K}|)$. Then

$$\text{Adv}_{rskp}^r(\mathcal{K}) = G_{q,\tau,k}(|\mathcal{K}|).$$

**Proof of Lemma 1.3.6.** Let $N = |\mathcal{K}|$. By Lemma 1.3.5, we have

$$\text{Adv}_{rskp}^r(\mathcal{K}) = \frac{1}{N} \sum_{x \in \mathcal{K}} \frac{(k - w(x))_{(\tau)}}{k_{(\tau)}} .$$

Since $B_{q,k}(r) \subseteq \mathcal{K} \subseteq B_{q,k}(r + 1)$. This means $B_{q,k}(i) \subseteq \mathcal{K}$ for $i = 0, \ldots, r$, and $\mathcal{K}$ contains
\[ N - B_{q,k}(r) \text{ vectors of weight } r + 1. \text{ Thus, the above equals} \]
\[
\frac{N - B_{q,k}(r)}{N} \frac{(k-r-1)\binom{k}{\tau}}{k!} + \frac{1}{N} \sum_{i=0}^{\tau} (q-1)^i \binom{k}{i} \frac{(k-i)\binom{k-1}{\tau}}{k!} = G_{q,k,\tau}(N)
\]
as claimed.

Next, we show that monotone sets sandwiched between Hamming balls are the extremal cases for the restricted subkey prediction game, meaning that they maximize the restricted subkey prediction advantage. The following is analogous to [11, Lemmas 6,8]. We streamline their analysis and extend it to large alphabets.

**Lemma 1.3.7** Let \( q,k,N \) be positive integers. Suppose \( q \geq 2 \), \( N \leq q^k \) and \( \tau \leq k \). Then, there is a non-empty monotone set \( \mathcal{K} \subseteq [q]^k \) of size \( N \) such that
\[
\text{Adv}_{q,k,\tau}^{rskp}(N) = \text{Adv}_{k,\tau}^{rskp}(\mathcal{K}) .
\]
Additionally, \( \mathcal{K} \) is also sandwiched between hamming balls, i.e. for \( r = rd_{q,k}(N) \),
\[
B_{q,k}(r) \subseteq \mathcal{K} \subset B_{q,k}(r+1) .
\]

The proof of Lemma [1.3.7] is deferred to Section [1.3.3]. As a direct corollary of Lemma [1.3.6] and Lemma [1.3.7] we get the following result.

**Corollary 1.3.8** Let \( q,\tau,k \) be positive integers such that \( \tau \leq k \) and \( q \geq 2 \). Then,
\[
\text{Adv}_{q,k,\tau}^{rskp}(N) = G_{q,k,\tau}(N) .
\] (1.13)

Hence, from this point on, we identify the two functions \( \text{Adv}_{q,k,\tau}^{rskp}(\cdot) \) and \( G_{q,k,\tau}(\cdot) \). Next, we observe a useful property of \( G_{q,k,\tau}(N) \). In particular, it is decreasing in the domain \([1..q^k]\).
Lemma 1.3.9 Let $q, \tau, k$ be positive integers such that $\tau \leq k$ and $q \geq 2$. Let $i, j$ be integers such that $1 \leq i \leq j \leq q^k$. Then,
\[ G_{q,k,\tau}(i) \geq G_{q,k,\tau}(j). \]

We proceed to relate the restricted subkey-prediction game to the subkey-prediction game via the lemma below.

Lemma 1.3.10 Let $\ell, q, k, \tau$ be integers such that $0 \leq \ell \leq k$, $q \geq 2$, and $1 \leq \tau \leq k$. Then,
\[ \text{Adv}^{\text{skp}}_{q,k,\tau}(\ell) \leq \text{Adv}^{\text{rskp}}_{q,k,\tau}(q^k - \ell). \]

The proofs of Lemma 1.3.9 and Lemma 1.3.10 are deferred to Section 1.3.3. Finally, we give a way to bound the expression $G_{q,k,\tau}(N)$. In particular, we show that it is at most the ratio of two hamming balls of the same radius $\text{rd}_{q,k}(N)$; one with dimension $k - \tau$ and one with dimension $k$. Recall that BKR did not give concrete numerical upper bounds for their subkey-prediction advantage, only estimates. Due to assuming the uniqueness of probes, we are able to simplify our expression $G_{q,k,\tau}(N)$. In particular, we note that for non-negative integers $k, i, \tau$ such that $i, \tau \leq k$,
\[ \binom{k}{i} \binom{k - i}{\tau} \frac{k(\tau)}{i(i)} \frac{(k - i)(\tau)}{i(i)} = \binom{k - \tau}{i}. \quad (1.14) \]

This property allows us to prove the following lemma.

Lemma 1.3.11 Let $N, q, k, \tau, r$ be positive integers such that $q \geq 2, N \leq q^k, \tau \leq k$ and $r \leq \text{rd}_{q,k}(N)$. Then
\[ G_{q,k,\tau}(N) \leq \frac{B_{q,k-\tau}(r)}{B_{q,k}(r)}. \]

Proof of Lemma 1.3.11: By Lemma 1.3.9,
\[ G_{q,k,\tau}(N) \leq G_{q,k,\tau}(B_{q,k}(r)). \]
By Equation (1.12) and Equation (1.14),

\[ G_{q,k,\tau}(B_{q,k}(r)) \leq \frac{1}{B_{q,k}(r)} \sum_{i=0}^{rd_{q,k}(B_{q,k}(r))} (q-1)^i \binom{k}{i} (k-i)_{(\tau)} \]

\[ = \frac{1}{B_{q,k}(r)} \sum_{i=0}^{r} (q-1)^i \binom{k-\tau}{i} \]

\[ = \frac{B_{q,k-\tau}(r)}{B_{q,k}(r)}. \]

The proof of Theorem 1.3.1 follows directly.

**Proof of Theorem 1.3.1:** Note that when \( r = 0 \), Equation (1.4) is trivially true. Hence, we let \( r \leq rd_{q,k}(N) \) be a positive integer. Then,

\[ \text{Adv}^{skp}_{q,k,\tau}(l) \leq \text{Adv}^{rskp}_{q,k,\tau}(q^{k-l}) \]

\[ = G_{q,k,\tau}(q^{k-l}) \]

\[ \leq \frac{B_{q,k-\tau}(r)}{B_{q,k}(r)}. \]

**Proof of Lemma 1.3.7**

Let

\[ \mathcal{T} = \left\{ \mathcal{K} \subseteq [q]^k : |\mathcal{K}| = N \text{ and } \text{Adv}^{rskp}_{k,\tau}(\mathcal{K}) = \text{Adv}^{rskp}_{q,k,\tau}(N) \right\}. \]

Let \( \mathcal{K} \in \mathcal{T} \) be the minimal weight element, i.e. the element \( \mathcal{K} \in \mathcal{T} \) that minimizes the value \( w(\mathcal{K}) = \sum_{x \in \mathcal{K}} w(x) \). We will show that \( \mathcal{K} \) is a set satisfying the properties claimed in the lemma. We will prove the two properties separately, namely that \( \mathcal{K} \) is monotone and \( B_{q,k}(r) \subseteq \mathcal{K} \subseteq B_{q,k}(r+1) \). We first claim that \( \mathcal{K} \) is monotone. The idea is to define a “shifting” operation for
any set $K' \subseteq [q]^k$ at a coordinate to increase $\text{Adv}^\text{rskp}_{k,\tau}(K')$ while decreasing $w(K')$. Seeking a contradiction, suppose $K$ is not monotone. Without loss of generality, suppose that for all pairs of $x \in K$ and $y \not\in K$ such that $y \leq x$, we have that $x$ and $y$ differ only in the first component. We build another set $K'$ with the following properties.

1. $|K'| = |K|
2. w(K') \leq w(K)
3. $\text{Adv}^\text{rskp}_{k,\tau}(K') \geq \text{Adv}^\text{rskp}_{k,\tau}(K)$

We first explain briefly explain the construction of $K'$ on the high level before giving the formal construction. Let $z \in [q]^{k-1}$. We will attempt to “swap” vectors of the form $\alpha \parallel z$, for $\alpha \in [q]$, in and out of $K$. The swapping is done in two cases. We define $D_z$ to contain the $\alpha$’s such that $\alpha \parallel z \in K$. First, if $0 \in D_z$ or $D_z = \emptyset$, no swapping will be done. Second, if $0 \not\in D_z$ and $D_z \neq \emptyset$, then we will do the following. Let $\beta = \max D_z$. We will remove the element $\beta \parallel z$ from $K$ and add the element $0 \parallel z$ to $K$. After such operations are done for all $z \in [q]^{k-1}$, the resulting set will be $K'$. Formally, the construction of $K'$ is given below. $K'$ is constructed from $K$ via the function $\phi : [q]^k \rightarrow [q]^k$, which is defined relative to the set $B$ (set $A$ is used in the later analysis). Sets $A$ and $B$ partition the set of strings of length $k - 1$. Set $A$ consists of $z$’s such that no swapping will be done. Set $B$ consists of $z$’s such that swapping will be done. The formal definition for $A, B, \phi$, 

and $\mathcal{K}'$ is as follows:

$$A = \left\{ z \in \mathbb{Z}_q^{k-1} : 0 \in D_z \text{ or } D_z = \emptyset \right\},$$

$$B = \left\{ z \in \mathbb{Z}_q^{k-1} : 0 \not\in D_z \text{ and } D_z \neq \emptyset \right\},$$

$$\phi(\alpha||z) = \begin{cases} 
0||z & \text{if } z \in B \text{ and } \alpha = \max D_z \\
(\max D_z)||z & \text{if } z \in B \text{ and } \alpha = 0 \\
\alpha||z & \text{otherwise} 
\end{cases},$$

$$\mathcal{K}' = \left\{ \phi(x) : x \in \mathcal{K} \right\}.$$

By construction, we note that the swapping operation preserves the size of the set and only decreases its overall weight. Hence, $|\mathcal{K}'| = |\mathcal{K}|$ and $w(\mathcal{K}') \leq w(\mathcal{K})$. It remains to show property (3). Let $A$ be an adversary such that $\text{Adv}^r_{\text{skp}}(A, \mathcal{K}) = \text{Adv}^r_{\text{skp}}(\mathcal{K})$. Consider the adversary $A'$ that behaves exactly as $A$ with the exception that it always guess 0 for the first position. More precisely, $A'$ does the following.

**Adversary $A'((s_1, \ldots, s_\tau))$**

$$J' \leftarrow A((s_1, \ldots, s_\tau))$$

For $i \leftarrow 1, \ldots, \tau$ do

If $s_i = 1$ then $J'[i] \leftarrow 0$

Return $J'$

Let $P(\cdot)$ denote the probability function in game $G^r_{\text{skp}}(A, \mathcal{K})$ and $P'(\cdot)$ the probability function in game $G^r_{\text{skp}}(A', \mathcal{K}')$. We now define three events for both games $G^r_{\text{skp}}(A, \mathcal{K})$, $G^r_{\text{skp}}(A', \mathcal{K}')$, where $z \in [q]^{k-1}$. 

28
WIN : The game returns true

ONE : \(1 \in \{s_1, \ldots, s_\tau\}\)

\(S_z : (K[1..k] = z), \) ONE and \((\forall i, s_i \neq 1 : J'[i] = K[s_i])\)

Note that \(P(\text{ONE}) = P'(\text{ONE})\), and \(P(\text{WIN} | \neg \text{ONE}) = P'(\text{WIN} | \neg \text{ONE})\). We claim that \(P(\text{WIN} | \text{ONE}) \leq P'(\text{WIN} | \text{ONE})\). If so we have

\[\text{Adv}_{k,\tau}^{\text{skp}}(\mathcal{A}, \mathcal{K}) = P(\text{WIN})\]

\[= P(\text{WIN} | \text{ONE}) \cdot P(\text{ONE}) + P(\text{WIN} | \neg \text{ONE}) \cdot P(\neg \text{ONE})\]

\[= P(\text{WIN} | \text{ONE}) \cdot P'(\text{ONE}) + P'(\text{WIN} | \neg \text{ONE}) \cdot P'(\neg \text{ONE})\]

\[\leq P'(\text{WIN} | \text{ONE}) \cdot P'(\text{ONE}) + P'(\text{WIN} | \neg \text{ONE}) \cdot P'(\neg \text{ONE})\]

\[= P'(\text{WIN}) = \text{Adv}_{k,\tau}^{\text{skp}}(\mathcal{A}', \mathcal{K}') .\]

So now we need to show that \(P(\text{WIN} | \text{ONE}) \leq P'(\text{WIN} | \text{ONE})\). We have

\[P(\text{WIN} | \text{ONE}) = \sum_{z \in [q]^{k-1}} P(\text{WIN} | S_z) \cdot P(S_z)\]

\[= \sum_{z \in [q]^{k-1}} P(\text{WIN} | S_z) \cdot P'(S_z) \quad (1.15)\]

\[\leq \sum_{z \in [q]^{k-1}} P'(\text{WIN} | S_z) \cdot P'(S_z) \quad (1.16)\]

\[= P'(\text{WIN} | \text{ONE}) .\]

Equation (1.15) is true because \(P(S_z) = P'(S_z)\) for all \(z \in [q]^{k-1}\), since the swapping operation does not change the last \(k - 1\) component of any vector. Next, we argue the validity of Equation (1.16).

Let \(z \in [q]^{k-1}\) such that \(P(S_z) \neq 0\) (and hence \(P'(S_z) \neq 0\)), which means that there is some \(\alpha \in [q]\) such that \(\alpha \| z \in \mathcal{K}\). For any \(z \in [q]^{k-1}\), consider the sets \(U_z = \{\alpha \in [q] : \alpha \| z \in \mathcal{K}\}\) and
$V_z = \{ \alpha \in [q] : \alpha \parallel z \in \mathcal{K} \}$. Note that $P(\text{WIN} | s_z) \leq 1/|U_z|$ and $P'(\text{WIN} | s_z) \leq 1/|V_z|$. Additionally, we note that $|U_z| = |V_z|$, and $V_z$ always contains 0. Since $\mathcal{A}'$ always guess 0 for the first component, we have $P'(\text{WIN} | s_z) = 1/|V_z|$. Therefore,

$$P(\text{WIN} | s_z) \leq \frac{1}{|U_z|} = \frac{1}{|V_z|} = P'(\text{WIN} | s_z).$$

Next, we show that $\mathcal{K}$ must be sandwiched between two hamming balls. We first claim that $B_{q,k}(r) \subseteq \mathcal{K}$. Seeking a contradiction, suppose that $B_{q,k}(r) \not\subseteq \mathcal{K}$. Let $x'$ be a point in $B_{q,k}(r) \setminus \mathcal{K}$ of minimal Hamming weight. Let $x$ be a point in $\mathcal{K} \setminus B_{q,k}(r)$ of maximal Hamming weight. We claim that $w(x) > w(x')$, otherwise $B_{q,k}(r) \subseteq \mathcal{K}$. Let $\mathcal{K}'$ be obtained by removing $x$ from $\mathcal{K}$ and then adding $x'$, i.e. $\mathcal{K}' = (\mathcal{K} \setminus \{x\}) \cup \{x'\}$. Because $x'$ was minimal in Hamming weight and $x$ was maximal in Hamming weight, the set $\mathcal{K}'$ continues to be monotone, and it has size $N$. Also $g_{k,\tau}(\mathcal{K}) < g_{k,\tau}(\mathcal{K}')$ because $w(x) > w(x')$. Hence, by Lemma 1.3.5

$$\text{Adv}^{\text{rskp}}_{k,\tau}(\mathcal{K}) = g_{k,\tau}(\mathcal{K}) < g_{k,\tau}(\mathcal{K}') = \text{Adv}^{\text{rskp}}_{q,k,\tau}(\mathcal{K}').$$

This contradicts the assumption that $\text{Adv}^{\text{rskp}}_{q,k,\tau}(N) = \text{Adv}^{\text{rskp}}_{k,\tau}(\mathcal{K})$. Hence, it must be that $B_{q,k}(r) \subseteq \mathcal{K}$. Now suppose $\mathcal{K} \not\subseteq B_{q,k}(r+1)$. Let $x'$ be a point in $B_{q,k}(r+1) \setminus \mathcal{K}$. Such a point exists because we know that $N < B_{q,k}(r+1)$. It must be that $w(x') = r+1$ since $B_{q,k}(r) \subseteq \mathcal{K}$. Let $x$ be a point in $\mathcal{K} \setminus B_{q,k}(r+1)$ of maximal Hamming weight. Note that $w(x) > r + 1 = w(x')$. Let $\mathcal{K}'$ be obtained by removing $x$ from $\mathcal{K}$ and then adding $x'$, meaning $\mathcal{K}' = (\mathcal{K} \setminus \{x\}) \cup \{x'\}$. The set $\mathcal{K}'$ continues to be monotone, and it has size $N$. Also $g_{k,\tau}(\mathcal{K}) < g_{k,\tau}(\mathcal{K}')$ because $w(x) > w(x')$. Hence, by Lemma 1.3.5

$$\text{Adv}^{\text{rskp}}_{k,\tau}(\mathcal{K}) = g_{k,\tau}(\mathcal{K}) < g_{k,\tau}(\mathcal{K}') = \text{Adv}^{\text{rskp}}_{k,\tau}(\mathcal{K}').$$

This contradicts the assumption that $\text{Adv}^{\text{rskp}}_{q,k,\tau}(N) = \text{Adv}^{\text{rskp}}_{k,\tau}(\mathcal{K})$. Hence, it must be that $\mathcal{K} \subseteq$
\( \text{B}_{q,k}(r + 1) \).

**Proof of Lemma 1.3.9 and Lemma 1.3.10**

To prove Lemma 1.3.9 and Lemma 1.3.10, we recall the notion of discrete concavity. Suppose \( F : [1..M] \to \mathbb{R} \). We say that \( F \) is concave if \( F(a + 1) - F(a) \leq F(b + 1) - F(b) \) for all \( a, b \in [1..M] \) satisfying \( a \geq b \). Now suppose \( t, m \) are integers with \( 1 \leq m \leq t \). Then we let

\[
S(M, m, t) = \left\{ (x_1, \ldots, x_m) \in [1..M]^m : x_1 + \cdots + x_m = t \right\}.
\]

Define \( F^m : [1..M]^m \to \mathbb{R} \) by \( F^m(x_1, \ldots, x_m) = F(x_1) + \cdots + F(x_m) \). We use the following lemma proved by [11].

**Lemma 1.3.12 ([11])** Suppose \( F : [1..M] \to \mathbb{R} \) is concave. Suppose \( 1 \leq m \leq t \) are integers such that \( m \) divides \( t \) and \( t/m \in [1..M] \). Then

\[
\max_{(x_1, \ldots, x_m) \in S(M, m, t)} F^m(x_1, \ldots, x_m) = m \cdot F(t/m).
\]

**Lemma 1.3.13** The function, \( F_{q,k,\tau} : [1..q^k] \to \mathbb{R} \), defined below, is concave.

\[
F_{q,k,\tau}(N) = \frac{N}{q^k} \cdot \text{Adv}^*_{q,k,\tau}(N).
\]

**Proof:** Let \( N_0, N_1 \) be two integers such that \( q^k \geq N_0 \geq N_1 \geq 1 \). Consider, for \( i = 0, 1 \),

\[
\Delta_i = F_{q,k,\tau}(N_i + 1) - F_{q,k,\tau}(N_i),
\]

For \( i = 0, 1 \), we let \( r_i \) be defined as follows. If \( N_i = B_{q,k}(r) \) for some \( r \), then we take \( r_i \) to be the value such that \( B_{q,k}(r_i) = N_i \). Otherwise, we let \( r_i = rd_{q,k}(N_i) + 1 \). Note that we can now express
\[ \Delta_i \text{ in terms of } r_i \text{ as follow (via Equation (1.12)),} \]

\[ \Delta_i = q^k \cdot \frac{(k - r_i)(\tau)}{k(\tau)}. \]

Since \( N_0 \geq N_1 \), we note that \( r_0 \geq r_1 \). Therefore, we have \( \Delta_0 \leq \Delta_1 \) and that \( F_{q,k,\tau} \) is concave. □

We first prove Lemma 1.3.9 using Lemma 1.3.13.

Proof of Lemma 1.3.9: Note that,

\[ G_{q,k,\tau}(N) = \frac{q^k \cdot F_{q,k,\tau}(N)}{N}. \]

We let \( \Delta_i = q^k \cdot F_{q,k,\tau}(i+1) - q^k \cdot F_{q,k,\tau}(i) \) for all \( i = 0, \ldots, q^k - 1 \). We define \( \Delta_0 = q^k \cdot F_{q,k,\tau}(1) = q^k \cdot G_{q,k,\tau}(1) \). Hence, by construction \( G_{q,k,\tau}(i) = (\sum_{j=0}^{i-1} \Delta_j) / i \). Note that since \( F_{q,k,\tau}(\cdot) \) is concave in the domain \([1..q^k]\), the sequence \( \Delta_1, \ldots, \Delta_{q^k-1} \) is non-increasing, meaning that \( \Delta_i \geq \Delta_j \) whenever \( 1 \leq i \leq j \leq q^k - 1 \). Additionally, we check that \( \Delta_0 = q^k \) and \( \Delta_1 \leq q^k \), hence \( \Delta_0 \geq \Delta_1 \). Therefore, the partial averages of the sequence \( \Delta_0, \ldots, \Delta_{q^k-1}, \)

\[ \left( \sum_{j=0}^{i-1} \Delta_j \right) / i = G_{q,k,\tau}(i), \]

is non-increasing as claimed. □

Lastly, we prove Lemma 1.3.10 using Lemma 1.3.12 and 1.3.13.

Proof of Lemma 1.3.10: Let \( M = q^k, m = q^\ell \) and \( t = q^k \). We note that the leakage function \( Lk : [q]^k \rightarrow [q]^\ell \) defines a partition of \([q]^k\) into \( q^\ell \) sets, with each set being \( Lk^{-1}(L) \) for some \( L \in [q]^\ell \). Hence, we can expand \( \Pr[\mathbf{G}_{q,k,\tau}^{skp}(\mathcal{A}, Lk)] \) by conditioning on the value of \( L \). Suppose
[\{q\}]^\ell = \{L_1, \ldots, L_m\}. We let \(N_i = |L_k^{-1}(L_i)|\). We derive

\[
\text{Adv}_{q,k,\tau}^{\text{skp}}(\ell) \\
= \max_{L_k} \left( \sum_L \frac{|L_k^{-1}(L)|}{q^k} \cdot \max_{\mathcal{A}} \Pr[\mathcal{G}_{q,k,\tau}^{\text{skp}}(\mathcal{A}, L_k) | L_k(K) = L] \right) \\
= \max_{L_k} \left( \sum_L \frac{|L_k^{-1}(L)|}{q^k} \cdot \text{Adv}_{k,\tau}^{\text{rskp}}(L_k^{-1}(L)) \right) \\
\leq \max_{(N_1, \ldots, N_m) \in \mathcal{S}(M, m, t)} \sum_{i=1}^m F_{q,k,\tau}(N_i) \\
= \max_{(N_1, \ldots, N_m) \in \mathcal{S}(M, m, t)} F_{q,k,\tau}^m(N_1, \ldots, N_m) \\
= m \cdot F_{q,k,\tau}(2^{k-\ell}) \\
= m \cdot \frac{q^{k-\ell}}{q^k} \cdot \text{Adv}_{q,k,\tau}^{\text{rskp}}(q^{k-\ell}) = \text{Adv}_{q,k,\tau}^{\text{rskp}}(q^{k-\ell}).
\] (1.17)

Equation (1.17) is justified since \(F_{q,k,\tau}\) is concave and \(t/m = 2^{k-\ell}\). Equation (1.18) is by definition of \(F\) and because \(m = q^\ell\). [1]

1.3.4 Multi-challenge Subkey Prediction

Here, we present an extension of \(\mathcal{G}_{q,k,\tau}^{\text{skp}}\) with multiple challenges, \(\mathcal{G}_{q,k,\tau,t}^{\text{mcskp}}\) (Fig. 1.3), the multi-challenge subkey prediction game. Note that [11] considers the multi-challenge version directly. However, we only need this extension in the proof of Theorem 1.4.1.

Let \(q, k, \tau, t, \ell\) be positive integers such that \(q \geq 2, k \geq \tau, k \geq \ell, t \geq 1\). We define the following advantages associated with the game \(\mathcal{G}_{q,k,\tau,t}^{\text{mcskp}}\), analogously to the advantages associated with \(\mathcal{G}_{q,k,\tau}^{\text{skp}}\).
\begin{figure}[h]
\centering
\begin{tabular}{|c|}
\hline
Game $G_{q,k,\tau,t}^{\text{mcskp}}(A, Lk)$ \\
$K \leftarrow [q]^k; L \leftarrow Lk(K)$ \\
For $i \in [t]$ do $p_i \leftarrow [k]^{(\tau)}$ \\
$J \leftarrow sA(L, p_0, \ldots, p_{t-1})$ \\
Return $(\exists i \in [t]: J = K[p_i])$
\hline
\end{tabular}
\caption{Multi-challenge subkey prediction game $G_{q,k,\tau,t}^{\text{mcskp}}$.}
\end{figure}

\begin{align*}
\text{Adv}_{q,k,\tau,t}^{\text{mcskp}}(A, Lk) &= \Pr\left[ G_{q,k,\tau,t}^{\text{mcskp}}(A, Lk) \right], \\
\text{Adv}_{q,k,\tau,t}^{\text{mcskp}}(Lk) &= \max_A \text{Adv}_{q,k,\tau,t}^{\text{skp}}(A, Lk), \\
\text{Adv}_{q,k,\tau,t}^{\text{mcskp}}(\ell) &= \max_{Lk : [q]^k \rightarrow [q]^\ell} \text{Adv}_{q,k,\tau,t}^{\text{mcskp}}(Lk).
\end{align*}

\textbf{Lemma 1.3.14} Let $q, k, \tau, t$ be positive integers such that $q \geq 2$, $k \geq \tau$, $k \geq \ell$, $t \geq 1$. Then,

\begin{align*}
\text{Adv}_{q,k,\tau,t}^{\text{mcskp}}(\ell) &\leq t \cdot \text{Adv}_{q,k,\tau}^{\text{skp}}(\ell).
\end{align*}

\textbf{Proof:} Let $Lk : [q]^k \rightarrow [q]^\ell$ be any leakage function. Let $A$ be a $G_{q,k,\tau,t}^{\text{mcskp}}$ adversary. We construct $G_{q,k,\tau}^{\text{skp}}$ adversary $A'$ such that

\begin{align*}
\text{Adv}_{q,k,\tau}^{\text{skp}}(A', Lk) &\geq \frac{1}{t} \cdot \text{Adv}_{q,k,\tau,t}^{\text{mcskp}}(A, Lk),
\end{align*}

which implies the lemma by maximizing over all $A$ and $Lk$. $A'$ is defined as follows.

\begin{align*}
\text{Adversary } A'(L, p') \\
j &\leftarrow [t]; p_j \leftarrow p' \\
\text{For } i \in [t] \setminus \{j\} &\text{ do } p_i \leftarrow [k]^{(\tau)} \\
J' &\leftarrow sA(L, p_0, \ldots, p_{t-1})
\end{align*}
Let \( E_1 \) be the event that \( \mathcal{A} \) succeed in the game \( G^{\text{skp}}_{q,k,\tau}(\mathcal{A}',\mathcal{L}k) \), i.e. \( J' = K[p_\alpha] \) for some \( \alpha \in [t] \). Note that \( \alpha \) is random variable that is well-defined given \( E_1 \) (in case \( J' = K[p_\alpha] \) for multiple \( \alpha \in [t] \), we can take the smallest one). We note that since \( \mathcal{A}' \) simulates the multi-challenge game for \( \mathcal{A} \) perfectly, \( \Pr[E_1] = \text{Adv}_{q,k,\tau,\tau}(\mathcal{A}') \). Let \( E_2 \subseteq E_1 \) be the event that \( \mathcal{A}' \) also guesses the correct \( \alpha \), i.e. \( j = \alpha \) in the game \( G^{\text{mskp}}_{q,k,\tau,\tau}(\mathcal{A}',\mathcal{L}k) \). We note that \( \Pr[E_2] = \frac{1}{t} \cdot \Pr[E_1] \), since \( j \) is independently uniform in \([t] \) and the distribution of \((p_0, \ldots, p_{r-1}) \) does not depend on the value of \( j \). Notice that \( \text{Adv}_{q,k,\tau}(\mathcal{A}',\mathcal{L}k) \geq \Pr[E_2] = \frac{1}{t} \cdot \Pr[E_1] = \frac{1}{t} \cdot \text{Adv}_{q,k,\tau,\tau}(\mathcal{A}',\mathcal{L}k) \). \( \blacksquare \)

### 1.4 Big-Key Symmetric Encryption

In [11], Big-Key symmetric encryption schemes are constructed modularly from Big-Key encapsulation schemes. In this section, we present a block-based big key encapsulation scheme.
that is more efficient than achieved previously.

**Key Encapsulation Schemes.** A (symmetric, Big-Key) encapsulation schemes, on input a big key $K$ and a random string $R$, returns a (short) key $K$. The string $R$ encapsulates the short key $K$ in the sense that any party holding the big key $K$ can derive $K$ from $R$. The security of a key encapsulation schemes is captured by $G^{\text{key}}_{\text{KEY}}(\mathcal{A})$ (Fig. 1.4). In this game, a big key $K$ is randomly sampled. The goal of the two-stage adversary $\mathcal{A}$ is to guess whether the real-or-random oracle, $\text{ROR}$, is returning real keys, derived using key encapsulation scheme $\text{KEY}$ from randomly sampled $R$, or randomly sampled keys that is independent of $R$. In its first stage, $\mathcal{A}$ gets access to $H$ and chooses a leakage function $L_k$ and state $\sigma$. Next, the game computes $L \leftarrow L_k^H(K)$ and run the second stage of $\mathcal{A}$ with inputs $L, \sigma$ and oracles $\text{ROR}$ and $H$. $\mathcal{A}$ wins the game if it successfully guesses the bit $b$. We define the following advantage of $\mathcal{A}$ against key encapsulation scheme $\text{KEY}$

$$\text{Adv}^{\text{key}}_{\text{KEY}}(\mathcal{A}) = 2 \cdot \Pr \left[ G^{\text{key}}_{\text{KEY}}(\mathcal{A}) \right] - 1.$$

**Our Construction.** Our random oracle model construction is given in Fig. 1.5.

**Theorem 1.4.1** Let $k, b, \kappa, \tau, r \geq 1$ be integers. Let $q = 2^b$. Let $\text{KEY} = \text{XKEY}_{q,k,\kappa,\tau,r}$ be the big-key encapsulation scheme associated to them as per Fig. 1.5. Let $\mathcal{A}$ be an adversary making at most $t$ queries to its $\text{ROR}$ oracle and leaking $\ell \cdot b$ bits. Assume the number of $H$ queries made by $\mathcal{A}$ in its first stage, plus the number made by the oracle leakage function $L_k$ that it outputs in this stage, is at most $q_1$, and the number of $H$ queries made by $\mathcal{A}$ in its second stage is at most $q_2$. Then

$$\text{Adv}^{\text{key}}_{\text{KEY}}(\mathcal{A}) \leq q_2 \cdot t \cdot \text{Adv}^{\text{skp}}_{q,k,\tau}(\ell) + \frac{t \cdot (2q_1 + t - 1)}{2r+1}.$$

(1.20)

The proof of Theorem 1.4.1 is deferred to Section 1.4.1.

**Sampling Unique Probes.** In $\text{XKEY}$, we have outsourced the sampling of the unique probes to the variable-range random oracle. We note that sampling from $[k]^{(\ell)}$ can be done via
Algorithm \( \text{SE}.\text{Enc}^H(K,M) \)
\[ R \leftarrow \{0,1\}^*; K \leftarrow \text{KEY}^H(K,R) \]
\[ C \leftarrow \text{AE.Enc}(K,M); \overline{C} \leftarrow (R,C) \]
Return \( \overline{C} \)

Algorithm \( \text{SE}.\text{Dec}^H(K,M) \)
\[ (R,C) \leftarrow \overline{C} \]
\[ K \leftarrow \text{KEY}^H(K,R) \]
\[ M \leftarrow \text{AE.Dec}(K,C) \]
Return \( M \)

**Figure 1.6:** Big-Key Symmetric Encryption Scheme [11, Section 5], \( \text{SE} \), using a standard symmetric key encryption scheme \( \text{AE} \) and a key encapsulation mechanism \( \text{KEY} \).

rejection sampling efficiently. For example, per Lemma 1.7.1 in Section 1.7 it holds with all but \( 2^{-3\tau} \) probability that \( 4\tau \) samples from \( [k] \) contains \( \tau \) unique probes (hence for parameters involved in Fig. 1.1 the failure probability is less than \( 2^{-129} \) since \( \tau \geq 43 \)).

**Symmetric Encryption Schemes.** To obtain a (big-key) symmetric encryption scheme, one can plug our \( XKEY \) construction directly into the (big-key) symmetric encryption scheme (in Fig. 1.6) by BKR. For security, we omit the details here and appeal to [11, Theorem 13].

**Efficiency.** Let \( k^* = 8 \cdot 10^{11} = 100 \) GBytes, and \( \ell^* = 10 \) GBytes. Using \( b = 8 \cdot 512 = 512 \) Bytes, our \( XKEY \) makes roughly the same number of \( H \) queries compared to [11] but makes significantly less access into the big key \( K \) (43 vs. 271, Fig. 1.1). In practical instantiations where \( K \) is stored on slow storage medium (e.g. hard disk), this translate to 6x improvement in efficiency.

### 1.4.1 Proof of Theorem [1.4.1]

**Proof of Proof (of Theorem 1.4.1):** Consider the games, \( G_{m_0}, \ldots, G_{m_3} \) defined in Fig. 1.7. Let \( \text{KEY} = XKEY_{q,k,K,\ell}. \) We note that game \( G_{m_0} \), with the boxed code included, simulates the game \( G_{\text{KEY}}^\text{key}(\mathcal{A}) \) exactly for \( b = 1 \) case and outputs true when \( \mathcal{A} \) outputs 1. Similarly, we note that \( G_{m_3} \), without the boxed code, simulates the game \( G_{\text{KEY}}^\text{key}(\mathcal{A}) \) exactly for \( b = 0 \) case and outputs true.
when $\mathcal{A}$ outputs 1. Hence,

$$\text{Adv}_{\text{key}}^{\text{key}}(\mathcal{A}) = \Pr[\text{Gm}_0] - \Pr[\text{Gm}_3].$$ (1.21)
### Game $G_{m_4}$

\[
\begin{align*}
K & \leftarrow [q]^k \\
\text{For } j & \leftarrow 1, \ldots, t \text{ do} \\
R[j] & \leftarrow \{0, 1\}^r ; \quad K[j] \leftarrow \{0, 1\}^k \\
P[j] & \leftarrow [k]^{(r)} \\
\text{stage} & \leftarrow 1; \quad (L_k, \sigma) \leftarrow \mathcal{A}_{\mathcal{H}^2}(\cdot); \quad L \leftarrow L_k^{\mathcal{H}2}(K) \\
\text{stage} & \leftarrow 2; \quad b' \leftarrow \mathcal{A}_{\mathcal{ROR}, \mathcal{H}^2}(\sigma, L) \\
\text{Return} & (b' = 1)
\end{align*}
\]

$ROR()$

\[
\begin{align*}
j & \leftarrow j + 1; \quad \text{Return} (R[j], K[j])
\end{align*}
\]

### Figure 1.8: Game $G_{m_4}$.

Note that $T_0$ is a table obtained via coin-fixing.

<table>
<thead>
<tr>
<th>Adversary $\mathcal{B}(L, pp_0, \ldots, pp_{t-1})$</th>
<th>$H_2(x, Img)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i \leftarrow 0; \text{For } j \in [t] \text{ do}$</td>
<td>If not $T_0[x, Img]$ then</td>
</tr>
<tr>
<td>$R[j] \leftarrow {0, 1}^r ; P[j] \leftarrow pp_j$</td>
<td>$T_0[x, Img] \leftarrow \text{Img}$</td>
</tr>
<tr>
<td>$b' \leftarrow \mathcal{A}_{\mathcal{ROR}, \mathcal{H}^2}(L)$</td>
<td>If stage = 2 then For $j \in [t]$ do</td>
</tr>
<tr>
<td>$\alpha \leftarrow [i]$</td>
<td>$\text{If } x = R[j] \text{ and } Img = [k]^{(r)} \text{ then}$</td>
</tr>
<tr>
<td>Return $J_\alpha$</td>
<td>$T_0[x, Img] \leftarrow P[j]$</td>
</tr>
<tr>
<td>$ROR()$</td>
<td>$\text{If } x = R[j] \parallel J[j] \text{ and } Img = {0, 1}^k \text{ then}$</td>
</tr>
<tr>
<td>$j \leftarrow j + 1; \text{Return} (R[j], K[j])$</td>
<td>$\text{bad} \leftarrow \text{true}$</td>
</tr>
</tbody>
</table>

| $H_3(x, Img)$ |
|--------------------------------------------------|---------------|

We will proceed to bound $\Pr[G_{m_0}]$. Note that $G_{m_0}$ and $G_{m_1}$ are identical-until-bad. Hence, via the Fundamental Lemma of Game Playing [19]

\[
\Pr[G_{m_0}] = \Pr[G_{m_1}] + (\Pr[G_{m_0}] - \Pr[G_{m_1}]) 
\]

\[
\leq \Pr[G_{m_1}] + \Pr[G_{m_1} \text{ sets bad}] . \tag{1.22}
\]

Next, we claim that

\[
\Pr[G_{m_1} \text{ sets bad}] \leq \frac{t(t-1)}{2r+1} + \frac{t \cdot q_1}{2^r} . \tag{1.23}
\]
First, there is at most $t(t-1)/2^r+1$ probability of collision in the $r$-bit values $R[1], \ldots, R[t]$ by the birthday bound. Next, $H_0$ sets bad only when stage $= 1$, and there are exactly $q_1$ $H_0$-queries when stage $= 1$. We note that each $H_0$-query when stage $= 1$ has at most $t/2^r$ probability of setting bad since there are at most $t$ distinct values for $R[1], \ldots, R[t]$.

We proceed to bound $\Pr[G_{m1}]$. We note that $G_{m2}$, with the boxed code included, is equivalent to $G_{m1}$. Furthermore, $G_{m3}$, without the boxed code, is identical to $G_{m2}$ until bad is set. Hence,

$$\Pr[G_{m1}] = \Pr[G_{m2}] = \Pr[G_{m3}] - (\Pr[G_{m2}] - \Pr[G_{m3}])$$

(1.24)

$$\leq \Pr[G_{m3}] + \Pr[G_{m3} \text{ sets bad}].$$

Lastly, we claim that

$$\Pr[G_{m3} \text{ sets bad}] \leq q_2 \cdot t \cdot \text{Adv}_{skp}^{skp}(\ell),$$

(1.25)

Notice that the theorem follows from Equations (1.21), (1.22), (1.23), (1.24), and (1.25). It remains to show Equation (1.25). The justification of Equation (1.25) involves two step. First, we argue that there is some fixing of the coins of $A, H_1, L_k$, which results in a deterministic leakage function $L_k'$, an adversary $A'$, and partial $H$ table $T_0$ such that

$$\Pr[G_{m3} \text{ sets bad}] \leq \Pr[G_{m4} \text{ sets bad}],$$

(1.26)

where $G_{m4}$ is given in Fig. 1.8. Next, we show that

$$\Pr[G_{m4} \text{ sets bad}] \leq q_2 \cdot \text{Adv}_{skp}^{mcskp}(B, L_k') \leq q_2 \cdot t \cdot \text{Adv}_{skp}^{skp}(\ell),$$

(1.27)

by constructing a multi-challenge subkey prediction adversary $B$, which is given in Fig. 1.9. $B$ will embed the probes given, $pp_0, \ldots, pp_{t-1}$ into the $H$ response and run $A'$. It will guess, at random, one of the $H$ queries of $A'$ of the form $(R||J, \{0,1\}^k)$. Hence, if $G_{m4}$ sets bad,
then with at least \( \frac{1}{q_2} \) probability, \( \mathcal{B} \) succeeds. The second part of Equation (1.27) follows from Lemma 1.3.14. This justifies Equation (1.25) and concludes the proof of the theorem. \( \square \)

### 1.5 Big-Key Identification

**Identification Schemes.** An identification scheme \( \text{ID} \) specifies the following:

- Via \( \text{prm} \leftarrow \text{ID.ParamGen} \), parameter generation algorithm \( \text{ID.ParamGen} \) generates parameter \( \text{prm} \), which is a common input to all other algorithms.
- Via \( (\text{sk}, \text{vk}, \text{hlp}) \leftarrow \text{ID.KeyGen}(\text{prm}) \), key generation algorithm \( \text{ID.KeyGen} \) is run by the prover to generate secret key \( \text{sk} \), corresponding verification key \( \text{vk} \) and a string \( \text{hlp} \) called the help string. The last is information that, conceptually, can be viewed as part of the public verification key \( \text{vk} \), meaning public and available to the adversary, but to keep the verification key small, \( \text{hlp} \) is stored by the prover along with \( \text{sk} \).
- Via \( (\text{com}, \text{st}) \leftarrow \text{ID.Com}(\text{prm}) \), commitment algorithm \( \text{ID.Com} \) is run by the prover to generate its first message \( \text{com} \), called the commitment, along with state information \( \text{st} \) that it saves.
- Via \( c \leftarrow \{0, 1\}^{\text{ID.Chl}} \), the verifier generates a random challenge \( c \) to return to the prover.
- Via \( z \leftarrow \text{ID.Rsp}(\text{prm}, \text{hlp}, \text{sk}, \text{st}, c) \), deterministic response algorithm \( \text{ID.Rsp} \) is run by the prover to generate its response \( z \).
- Via \( d \leftarrow \text{ID.Vrf}(\text{prm}, \text{vk}, \text{com}, c, z) \), deterministic decision algorithm \( \text{ID.Vrf} \) returns a boolean decision \( d \) for the verifier to accept or reject.

In the ROM, algorithms may have oracle access to the random oracle \( H \). This syntax is non-asymptotic, in that there is no explicit security parameter. Correctness requires that

\[
\Pr[\text{Execute}_{\text{ID}}(\text{prm}, \text{vk}, \text{sk}, \text{hlp})] = 1
\]

for all \( \text{prm} \in [\text{ID.ParamGen}] \) and \( (\text{sk}, \text{vk}, \text{hlp}) \in [\text{ID.KeyGen}(\text{prm})] \), where
Game \( \mathcal{G}_{\text{ID}}^{\text{imp}}(\mathcal{A}) \)

\[
\begin{align*}
\text{prm} &\leftarrow \text{ID}.\text{ParamGen}; s \leftarrow 0 \\
(sk, vk, hlp) &\leftarrow \text{ID}.\text{KeyGen}(\text{prm}) \\
st &\leftarrow \mathcal{A}.\text{Setup}^{\text{Leak}_{k}, \text{Prover}, \text{H}}(\text{prm}, vk, hlp) \\
\text{com}, st' &\leftarrow \mathcal{A}.\text{Com}^{H}(st); \\
c &\leftarrow \{0, 1\}^{\text{ID}, \text{Chl}} \\
z &\leftarrow \mathcal{A}.\text{Rsp}^{H}(\text{prm}, hlp, sk, st', c) \\
d &\leftarrow \text{ID}.\text{Vrf}^{H}(\text{prm}, vk, \text{com}, c, z) \\
\text{Return } d
\end{align*}
\]

\(\text{Leak}_{k}(f)\)

\[
L \leftarrow f(sk); s \leftarrow s + |L|
\]

If \(s \leq \ell\) then return \(L\) else return \(\perp\)

---

Prover\((i, \text{args})\)

If \(\text{pst}[i] = \perp\) then / Commit

\[
(\text{pcom}[i], \text{pst}[i]) \leftarrow \text{ID}.\text{Com}(\text{prm}) \\
\text{Return } \text{pcom}[i]
\]

If \(\text{prsp}[i] = \perp\) then / Response

\[
\text{prsp}[i] \leftarrow \text{ID}.\text{Rsp}^{H}(\text{prm}, hlp, sk, \text{pst}[i], \text{args}) \\
\text{Return } \text{prsp}[i]
\]

Return \(\perp\)

\[
\text{H}(x, \text{Img})
\]

If \(T[x, \text{Img}] = \perp\) then \(T[x, \text{Img}] \leftarrow \text{Img} \\
\text{Return } T[x, \text{Img}]
\]

---

**Figure 1.10:** Game defining security of identification scheme \(\text{ID}\) under pre-impersonation leakage.

---

**Security of Identification Schemes.** We give definitions allowing concrete-security assessments. The core definition is that of adversary advantage. The notion captured is security against impersonation under active attack \([45, 15]\) in the further presence of leakage on the secret key \([6]\).

Let \(\text{ID}\) be an identification scheme. Let \(\ell\) be an integer representing a bound (in bits) on the leakage. Let \(\mathcal{A}\) be an impersonation adversary, made up of component algorithms \(\mathcal{A}.\text{Setup}, \mathcal{A}.\text{Com}, \text{and } \mathcal{A}.\text{Rsp}\). We associate to these the game of Fig. 1.10. First, the parameters and keys

---
are generated. Next, $\mathcal{A}$.Setup is run with access to a leakage oracle $\text{Leak}_\ell$ a prover oracle $\text{Prover}$ and the random oracle $H$. The leakage oracle takes input a function $Lk$ from the adversary and returns leakage $L = Lk(sk)$. This oracle can be called adaptively and any number of times, its code ensuring that the total number of bits returned to the adversary does not exceed $\ell$. The prover oracle allows an active attack in which the adversary, playing the role of a dishonest verifier, can generate prover instances and interact with them. The commitment and state of instance $i$ are produced by the game and stored as $\text{pcom}[i]$ and $\text{pst}[i]$, respectively. If instance $i$ has been activated, meaning $\text{pst}[i] \neq \perp$, then the adversary can submit, via $\text{args}$, a challenge of its choice, and obtain response $\text{prsp}[i]$. After exiting this setup phase, the adversary turns into a dishonest prover, aiming to convince the honest verifier to accept. It produces its commitment via $\mathcal{A}.\text{Com}$, receives a random challenge $c$, and produces its response via $\mathcal{A}.\text{Rsp}$. The game returns the boolean decision $d$ of the verifier’s decision function. We define the leakage impersonation advantage of $\mathcal{A}$ against ID to be

$$\text{Adv}_{\text{ID,}\ell}(\mathcal{A}) = \Pr \left[ \mathcal{G}_{\text{ID,}\ell}(\mathcal{A}) \right].$$

**GROUPS.** We fix a *bilinear group description* $G = (G, G_T, g, e, p)$, where

- $p \geq 3$ is a prime number that will be the order of the groups
- $G, G_T$ are (cyclic) groups of order $p$
- $g \in G$ is a generator of $G$
- $e : G \times G \to G_T$ is an efficiently computable, non-degenerate bilinear map. This means that (1) $e(g^a, g^b) = e(g, g)^{ab}$ for all $a, b \in [p]$, and (2) $e(g, g)$ is not the identity element of $G_T$.

We will base security on the assumed hardness of the CDH (Computational Diffie-Hellman) and DL (Discrete Logarithm) problems in $G$. The definitions are based on games $G^{\text{cdh}}$ and DL in
<table>
<thead>
<tr>
<th>Game $G_{G}^{cdh}(\mathcal{A})$</th>
<th>Game $G_{p,m,k,\tau}^{pskp}(\mathcal{A}, Lk)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(G, G_T, g, e, p) \leftarrow G$</td>
<td>For $i \in [k]$ do $sk[i] \leftarrow \mathbb{Z}_p^m$</td>
</tr>
<tr>
<td>$x, y \leftarrow [p]$; $h \leftarrow \mathcal{A}(G, g^x, g^y)$</td>
<td>$pp \leftarrow [k]^{\tau}$; $e \leftarrow \mathbb{Z}_p$</td>
</tr>
<tr>
<td>Return $(h = g^{xy})$</td>
<td>For $j \in [m]$ do $sk^* [j] = \sum_{i=0}^{\tau-1} (sk[pp][i])[j]e^i$</td>
</tr>
<tr>
<td></td>
<td>$L \leftarrow Lk(sk)$; $sk^* \leftarrow \mathcal{A}(pp, e, L)$</td>
</tr>
<tr>
<td></td>
<td>Return $(sk^* = sk)$</td>
</tr>
</tbody>
</table>

**Figure 1.11**: Left: Games $G_{G}^{cdh}$ and $DL_{G}$ defining the security of CDH and DL problems in $G$.

Right: Game $G_{p,m,k,\tau}^{pskp}(\mathcal{A}, Lk)$. Where $Lk : [q]^k \rightarrow [q]^\ell$ is a leakage function. $[k]^{\tau}$ contains the set of $\tau$-dimensional vectors over $[k]$ with distinct entries.

Fig. 1.11 associated to $G$ and an adversary $\mathcal{A}$. We define the following CDH and DL advantages:

$$\text{Adv}_{G}^{cdh}(\mathcal{A}) = \Pr[G_{G}^{cdh}(\mathcal{A})]$$

$$\text{Adv}_{G}^{dl}(\mathcal{A}) = \Pr[DL_{G}(\mathcal{A})].$$

Hardness of CDH of course implies hardness of DL. Quantitatively, given $\mathcal{A}$, one can construct $\mathcal{A}'$ with similar running time such that

$$\text{Adv}_{G}^{dl}(\mathcal{A}) \leq \text{Adv}_{G}^{cdh}(\mathcal{A}').$$

**ADW IDENTIFICATION SCHEME.** We present a variant of ADW’s identification scheme [6], which uses a random oracle to derive the challenges (as considered in [6] without analysis). The scheme $\text{ID} = \text{ADW}[G, k, m, \tau, r]$ is parameterized by a bilinear group description $G$ and positive integers $k, m, \tau, r$. We require that $m \geq 2$ and $k \geq \tau \geq 1$. Here $k$ is the number of blocks of the secret key, where each block is an $m$-dimensional vector over $\mathbb{Z}_p$, and $\tau$ is the number of probes that algorithms make into the secret key. The parameter $r$ determines the challenge length, meaning we set $\text{ID}.c = r$. The algorithms $\text{ID}.\text{ParamGen}, \text{ID}.\text{KeyGen}, \text{ID}.\text{Com}, \text{ID}.\text{Rsp}, \text{ID}.\text{Vrf}$ are
ID.\text{ParamGen}()  
\text{For } i \in [m] \text{ do } g_i \leftarrow G  
\text{Return } (g_0, \ldots, g_{m-1})

ID.\text{Com}(\text{prm})  
y \leftarrow (\mathbb{Z}_p)^m  
a \leftarrow \prod_{j=0}^{m-1} g_{[j]}^{y_{[j]}}  
\text{Return } (a, y)

Derive^H(R)  
\text{pp } \leftarrow H(R, [k])  
e \leftarrow H(0||R, [p]); c^* \leftarrow H(1||R, [p])  
\text{Return } (\text{pp}, e, c^*)

ID.\text{KeyGen}^H(\text{prm})  
s \leftarrow \mathbb{Z}_p^\star; \text{vk } \leftarrow g^s  
\text{For } i \in [k] \text{ do }  
\text{sk}[i] \leftarrow (\mathbb{Z}_p)^m  
pk[i] \leftarrow \prod_{j=0}^{m-1} g_{[i]}^{sk[i][j]}  
\sigma[i] \leftarrow (H(i, G)pk[i])^s  
\text{hlp } \leftarrow (pk, \sigma)  
\text{Return } (\text{sk}, \text{vk}, \text{hlp})

ID.\text{Rsp}^H(\text{prm}, \text{hlp}, \text{sk}, \text{st}, c)  
(\text{pp}, e, c^*) \leftarrow \text{Derive}^H(c)  
\text{For } j \in [m] \text{ do }  
\text{sk}^*[j] \leftarrow \sum_{i=0}^{s} \text{sk}[pp[i][j]] \cdot e^{i}  
pk^* \leftarrow \prod_{i=0}^{r-1} pk[pp[i]] e^{c}  
\sigma^* \leftarrow \prod_{i=0}^{r-1} \sigma[pp[i]] e^{c}  
\text{For } j \in [m] \text{ do }  
z \leftarrow y_j + e^c \cdot sk^*[j]  
\text{Return } (pk^*, \sigma^*, z)

ID.\text{Vrf}^H(\text{prm}, \text{vk}, \text{com}, c, z)  
a \leftarrow \text{com}  
(\text{pp}, e, c^*) \leftarrow \text{Derive}^H(c)  
(pk^*, \sigma^*, z) \leftarrow z  
A \leftarrow (\prod_{i=0}^{r-1} g_i^{z}) = a(pk^*)^{c^*}  
h_1 \leftarrow e(pk^* \prod_{i=0}^{r-1} H(pp[i], G))^c \cdot \text{vk}  
h_2 \leftarrow e(\sigma^*, g)  
B \leftarrow (h_1 = h_2)  
\text{Return } (A \wedge B)

**Figure 1.12:** Algorithms of identification scheme ID = ADW[$G, k, m, \tau, r$] associated to bilinear group description $G = (G, G_T, g, e, p)$ and parameters $k, m, \tau, r$ satisfying $m \geq 2$ and $k \geq \tau \geq 1$. Here $H$ is a variable range function, meaning $H(\cdot, \text{Img})$ returns outputs in the set (described by Img). In addition, algorithms KeyGen, Com, Rsp, Vrf also takes pm as argument.

Intuitively, the scheme consists of $k$ generalized Okamoto identification scheme [74, 6], and one instance of BLS signature scheme [28]. Each block of the secret key (in $\mathbb{Z}_p^m$) is a secret key for a generalized Okamoto identification scheme of dimension $m$. The public keys, $pk[0], \ldots, pk[k-1]$, of the $k$ Okamoto’s identification schemes, are signed using the BLS signature scheme under signing key $s$, yielding signatures $\sigma[0], \ldots, \sigma[k-1]$. The public verification key of the identification scheme, consists only of the verification key, $\text{vk}$, of the BLS signature scheme. During identification, a random $\tau$ instances out of $k$ instances is chosen (via $H$ by the verifier) and compressed via polynomial evaluation to $sk^*, pk^*$, and $\sigma^*$ by the prover. During response phase, the prover, in addition to answering the challenge from the Okamoto identification scheme, needs to transmit $pk^*$ and $\sigma^*$ to the verifier. We note that the signing key, $s$, of the underlying signature scheme must not be visible to the attacker. This signing key is simply be discarded after KeyGen. (However, we note that, as ADW has pointed out, there are advanced uses of this key such as
updating the big secret key.) The correctness of $ID = ADW[G, k, m, \tau, r]$ is checked as follows. Let $prm \in [ID.\text{ParamGen}]$ and $(sk, vk, hlp) \in [ID.\text{KeyGen}(prm)]$. We claim that, during a honest execution of the protocol $(\text{Execute}_{ID}(prm, sk, vk, hlp))$, the flags $A, B$ in $ID.\text{Vrf}$ will both be set to true. $A$ is set to true because

$$\prod_{i=0}^{m-1} g_i^{z[i]} = \prod_{i=0}^{m-1} g^{y[i] + c \cdot sk^*[i]} = \prod_{i=0}^{m-1} g^{y[i]} \cdot (\prod_{i=0}^{m-1} g^{sk^*[i]})^c = a \cdot pk^*c^* .$$

$B$ is set to true because

$$e(pk^* \prod_{i=0}^{\tau-1} H(pp[i], G)^e^i, vk) = e(\prod_{i=0}^{\tau-1} pk[pp[i]]^e_i \prod_{i=0}^{\tau-1} H(pp[i], G)^e_i, g) = e(\prod_{i=0}^{\tau-1} (pk[pp[i]]H(pp[i], G))^{e_i}, g) = e(\prod_{i=0}^{\tau-1} (\sigma[pp[i]])^{e_i}, g) = e(\sigma^*, g) .$$

Hence, $Pr[\text{Execute}_{ID}(prm, sk, vk, hlp)] = 1$, and $ID$ satisfies correctness.

**Efficiency.** As pointed out in [6], the identification scheme has nice efficiency properties. First, the public key (verification key) is very short (one group element). Second, the communication costs of all phases are very small. The bulk of communication happens in the response phase, which outputs 2 group elements and $m$ elements from $\mathbb{Z}_p$. Third, the scheme has probe complexity depending on $\tau$, which can be made small while preserving security. In particular, during each run of the protocol, only $\tau$ locations of the secret-key will be accessed (each location consist of $m$ elements of $\mathbb{Z}_p$). Fig. [1.2] demonstrates the computation and communication costs of different operations. Note that very small values of $\tau$ makes the scheme insecure. The crux of the
Table 1.2: Left: Table illustrating computation and communication cost of different operations of the identification scheme ADW\textsubscript{G,k,m,τ}. Chl here represents the challenge phase of the protocol. Right: Example parameters for ADW scheme to achieve 128-bit security. The schemes uses group of size \( p \) such that \( 2^{511} < p < 2^{512} \), and we impose a bound on the leakage of 10\% on a big-key of size 100 GB = \( 8 \times 10^{11} \) bits. For each value of \( m \) on the left column, we look the value of \( τ \) needed to achieve 128-bit security for the identification scheme, both using our bound and using ADW’s bound.

<table>
<thead>
<tr>
<th>Computation cost</th>
<th>KeyGen</th>
<th>Com</th>
<th>Chl</th>
<th>Rsp</th>
<th>Vrf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mult ( G )</td>
<td>( k \cdot m )</td>
<td>( m - 1 )</td>
<td>0</td>
<td>( 2τ )</td>
<td>( m + τ )</td>
</tr>
<tr>
<td>Exp ( G )</td>
<td>( k(m + 1) + 1 )</td>
<td>( m )</td>
<td>0</td>
<td>( 2τ - 2 )</td>
<td>( τ + m + 1 )</td>
</tr>
<tr>
<td>Mult ( Z_p )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( m )</td>
<td>1</td>
</tr>
<tr>
<td>Exp ( Z_p )</td>
<td>( m )</td>
<td>0</td>
<td>0</td>
<td>( 2τ )</td>
<td>( τ )</td>
</tr>
<tr>
<td>( e ) eval</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Communication cost</th>
<th>( G )</th>
<th>( Z_p )</th>
<th>( {0,1}^r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G )</td>
<td>-</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( Z_p )</td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( {0,1}^r )</td>
<td>-</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Security analysis amounts to giving a lower bound of \( τ \) for a desired security level. Here is where we make significant concrete security improvements over ADW.

**Concrete-security analysis.** Before we present the theorem stating the concrete security of the ADW identification scheme, we first need to define the following special subkey prediction game. The game \( G_{p,m,k,τ}^{pskp}(Lk, \mathcal{A}) \) (Fig. 1.11) captures a particular type of subkey prediction game in which the subkey is interpreted as a tuple of polynomials. In this game, the adversary \( \mathcal{A} \) needs to predict the value of these polynomials at a random point \( e \), which is given to \( \mathcal{A} \). We define the following prediction advantage

\[
\text{Adv}_{p,m,k,τ}^{pskp}(\ell) = \max_{\mathcal{A}, Lk: (Z_p^m)^k \rightarrow (Z_p^m)\ell} \text{Pr}[G_{p,m,k,τ}^{pskp}(\mathcal{A}, Lk)].
\]

We state a theorem which captures the concrete security of the ADW identification scheme. The theorem streamlines the original analysis of ADW to a precise relation of advantages, which allows us to instantiate parameters of practical sizes.
Theorem 1.5.1  Let $G = (G, G_T, g, e, p)$ be a group with efficient pairings. Let $ID = ADW_{G, k, m, \tau, r}$ be the ADW identification scheme shown in Fig. 1.12. Let $A = (A.\text{Setup}, A.\text{Com}, A.\text{Rsp})$ be a leakage impersonation adversary. Let $q$ denote the number of H queries plus the number of Prover queries that $A.\text{Setup}$ and $A.\text{Com}$ makes. Fig. 1.14 and Fig. 1.14 gives two adversaries $A_{\text{cdh}}$ and $A_{\text{dl}}$ such that

$$\text{Adv}_{\text{imp}}^{\text{id}, \ell}(A)^2 \leq \text{Adv}_{G}^{\text{cdh}}(A_{\text{cdh}}) + m \cdot \text{Adv}_{G}^{\text{dl}}(A_{\text{dl}}) + \text{Adv}_{p, m, k, \tau}^{\text{pskp}}(\ell + k/m) + q + 1/p.$$  \hspace{1cm} (1.28)

Additionally, let $t_1$ be the running time of $A.\text{Setup}$, $t_2$ be the running time of $A.\text{Com}$, $t_3$ be the running time of $A.\text{Rsp}$, and let $t_4$ be the running time of $ID.\text{KeyGen}$. We have that the running time of $A_{\text{cdh}}$ and $A_{\text{dl}}$ is approximately $t_1 + t_2 + 2 \cdot t_3 + t_4$.

The proof of Theorem 1.5.1 is given in Section 1.5.1. The following lemma relates $\text{Adv}_{p, m, k, \tau}^{\text{pskp}}(\ell + k/m)$ to the large-alphabet subkey prediction advantage (as bounded in Section 1.3.3).

Lemma 1.5.2  Let $p, m, k, \tau, \ell$ be positive integers, then

$$\text{Adv}_{p, m, k, \tau}^{\text{pskp}}(\ell) \leq \sqrt{\text{Adv}_{p, m, k, \tau}^{\text{skp}}(\ell)} + \frac{\tau}{p}.$$  

We note that with Lemma 1.5.2 we can bound the term $\text{Adv}_{p, k, k, \tau}^{\text{pskp}}(\ell)$ for any value $p, m, k, \tau, \ell$. Hence, the only term that is not explicitly bounded on the right-hand side of Equation (1.28) are $\text{Adv}_{G}^{\text{cdh}}(A)$ and $m \cdot \text{Adv}_{G}^{\text{cdh}}(A)$, which can be assumed to be small when the CDH and DL problems are suspected to be hard in group $G$.

Comparison with ADW’s analysis. Our analysis of ADW’s identification scheme improves upon the original analysis in the following ways. First, we analyze the scheme in which the challenge is generated using a random oracle directly. (The construction that uses a random oracle to derive the challenge is mentioned to be secure in [6] with no proof.) Sec-
ond, while ADW’s analysis is offered in the asymptotic case, we state and prove a reduction that gives concrete security, which lead to practical instantiation of parameters. The reduction gives a bound of the impersonation advantage in terms of three dominating quantities: CDH and DL advantages in $G$, and a special form of subkey-prediction advantage under polynomial compression, $\text{Adv}_{p,m,k,\tau}^{\text{pskp}}(\mathcal{L}_k, \mathcal{A})$. Hence, giving a good numerical bound of the impersonation advantage amounts to bounding $\text{Adv}_{p,m,k,\tau}^{\text{pskp}}(\mathcal{L}_k, \mathcal{A})$. Here is where we make significant improvements: we use the large-alphabet subkey prediction lemma (Theorem 1.3.1) as well as a tighter polynomial-evaluation entropy preservation lemma (Lemma 1.5.2) to give significantly better concrete bounds. The comparison of parameters can be found in Fig. 1.2.

**PARAMETER INSTANTIATION.** We give an example instantiation of the ADW identification scheme with 128-bits security. First, we find a pairing friendly group $G$ with symmetric pairing $\mathbf{e} : G \times G \to G_T$. Because of the square-root loss of security, we need 256-bit of security for CDH and DL in $G$. Hence, $G$ needs to be of size roughly 512 bits. We consider $\mathcal{G} = (G, G_T, g, \mathbf{e}, p)$, where $p$ is a prime of roughly 512 bits ($2^{511} < p < 2^{512}$). We represent elements in $\mathbb{Z}_p$ using exactly 512 bits. We pick a big key size of 100 GB, i.e. $k^* = 8 \cdot 10^{11}$. For a choice of $m \geq 2$, we have that the block size in bits is $b = m \cdot 512$. We let $k = k^* / b$ be the size of the big key in blocks. We fix a leakage rate of 10%. By Theorem 1.5.1 and Lemma 1.5.2 to achieve 128-bit security for the identification scheme, we need 512 bits of security from $\text{Adv}_{p,m,k,\tau}^{\text{pskp}}(\mathcal{L} + \frac{k}{m})$. Hence, we need

$$\tau = \text{Probes}_{k^*, \ell^* + k^*/m, s}(m \cdot 512)$$

probes. Values of $\text{Probes}_{k^*, \ell^* + k^*/m, s}(m \cdot 512)$ versus various values of $m$ is shown in Fig. 1.2 using both our bound and ADW’s bound.

**ENTROPY PRESERVATION UNDER POLYNOMIAL EVALUATION.** Lemma 1.5.2 relates the prediction advantage to the large-alphabet subkey prediction advantage. Note that our bound is
quantitatively better than \[6, \text{Corollary A.1}\]. In particular, we prove \(\frac{1}{2}\) rate entropy preservation while ADW proves a rate of \(\frac{1}{3}\). Before proving the lemma, we define the following quantities for jointly distributed random variables \((X, Y)\). Let \(X\) be a random variable, the prediction and collision probability of \(X\) is defined, respectively, to be

\[
\text{Pred}(X) = \max_x \Pr[X = x], \quad \text{CP}(X) = \Pr[X = X'],
\]

where \(X'\) is an independent random variable that is identically distributed to \(X\). Additionally, suppose that \((X, Y)\) are jointly distributed, we define the conditional prediction and collision probability of \(X\) given \(Y\), respectively, to be

\[
\widetilde{\text{Pred}}(X \mid Y) = \mathbb{E}_Y[\text{Pred}(X \mid Y)], \quad \widetilde{\text{CP}}(X \mid Y) = \mathbb{E}_Y[\text{CP}(X \mid Y)].
\]

We note that \(\text{Pred}(X \mid Y)\) and \(\text{CP}(X \mid Y)\) are random variables in \(Y\). We need the following well-known lemma,

**Lemma 1.5.3** Let \((X, Y)\) be jointly distributed random variables, then

\[
\widetilde{\text{CP}}(X \mid Y) \leq \widetilde{\text{Pred}}(X \mid Y) \leq \sqrt{\text{CP}(X \mid Y)}.
\]

**Proof:** For each value \(y\) of the random variable \(Y\), we consider the probability mass function of the random variable \(X \mid Y = y\), \(P_{X \mid Y = y}(\cdot)\). We note that

\[
\text{Pred}(X \mid Y = y) = \max_x P_{X \mid Y = y}(x),
\]

\[
\text{CP}(X \mid Y = y) = \sum_x P_{X \mid Y = y}(x)^2.
\]
First, we derive that 

$$\text{CP}(X \mid Y = y) \geq \left( \max_x P_{X \mid Y = y}(x) \right) \cdot \sum_x P_{X \mid Y = y}(x) = \text{Pred}(X \mid Y = y).$$

Taking expectation over $y$ sampled from $Y$ on both sides of the above equation, we obtain that 

$$\tilde{\text{Pred}}(X \mid Y) \leq \tilde{\text{CP}}(X \mid Y).$$

Next, we note that viewing $\text{Pred}(X \mid Y = y)$, $\sqrt{\text{CP}(X \mid Y = y)}$ as 1- and 2-norms of $P_{X \mid Y = y}(\cdot)$ respectively, we have $\text{Pred}(X \mid Y = y) \leq \sqrt{\text{CP}(X \mid Y = y)}$. Hence, by the above property and Jensen’s inequality

$$\tilde{\text{Pred}}(X \mid Y) = E_Y[\text{Pred}(X \mid Y)]$$

$$\leq E_Y[\sqrt{\text{CP}(X \mid Y)}]$$

$$\leq \sqrt{E_Y[\text{CP}(X \mid Y)]}$$

$$= \tilde{\text{CP}}(X \mid Y).$$

**Proof of Lemma 1.5.2** Let $\mathcal{A}$ be any adversary and $Lk : [q]^k \rightarrow [q]^\ell$ be a leakage function.

Consider the sample space defined by the experiment $G_{\text{pskp}}^{p,m,k,\tau}(\mathcal{A}, Lk)$ (all the coins used by the experiment and adversary $\mathcal{A}$). We consider all the variables used inside $G_{\text{pskp}}^{p,m,k,\tau}(\mathcal{A}, Lk)$ as random variables (e.g. $sk^*$ and $L = Lk(K)$). We note that

$$\Pr[G_{p,m,k,\tau}^{\text{pskp}}(\mathcal{A}, Lk)] \leq \tilde{\text{Pred}}(sk^* \mid pp, L, e).$$

Furthermore, by Lemma 1.5.3

$$\tilde{\text{Pred}}(sk^* \mid pp, L, e) \leq \sqrt{\tilde{\text{CP}}(sk^* \mid pp, L, e)}.$$

We now need to bound $\tilde{\text{CP}}(sk^* \mid pp, L, e)$. To compute this quantity. We consider another
independent execution of \( G_{p, m, k, \tau}^{pskp}(A, L_k) \), where the variables in the second execution is denoted with \( ' \), e.g. \( sk' \). We restrict to the event that \( L_k(sk) = L_k(sk') \) and \( pp = pp' \). We define polynomials \( p_1, \ldots, p_j \), which are functions of \( sk, sk', pp \), \( p_j(x) = \sum_{i=0}^{\tau-1} (sk[pp][i][j] - sk'[pp][i][j]) x^i \). Notice that these polynomials are of degree at most \( \tau \). If \( sk \neq sk' \), then at least one of \( p_j \) is a non-zero polynomial, and has at most \( \tau \) roots. Hence, if \( sk \neq sk' \), over a independently uniform \( e \), the probability that \( p_j(e) = 0 \) is at most \( \frac{\tau}{p} \) when \( p_j \) is not the zero polynomial. Finally, we derive that

\[
\tilde{\text{CP}}(sk^* \mid pp, L, e) = E_{pp, L, e} \left[ \text{CP}(sk^* \mid pp, L, e) \right] \\
\leq E_{pp, L, e} \left[ \Pr[sk[pp] = sk'[pp] \mid pp, L, e] \\
+ \Pr[\forall j \in [m] : p_j(e) = 0 \mid sk[pp] \neq sk'[pp], pp, L, e] \right] \\
= \tilde{\text{CP}}(sk[pp] \mid pp, L) + E_e \left[ \Pr[\forall j \in [m] : p_j(e) = 0] \right] \\
\leq \tilde{\text{Pred}}(sk[pp] \mid pp, L) + \frac{\tau}{p} \\
\leq \text{Adv}_{pskp}^{skp}(\ell) + \frac{\tau}{p}.
\]

1.5.1 Proof of Theorem 1.5.1

We follow the proof technique used by [6]. Let \( \text{ID} = \text{ADW}_{G, k, m, \tau, r} \) be the ADW identification scheme. The reduction is very similar to the reduction from [6] Appendix B.5. Rewind attempts to run a given leakage impersonation adversary \( A \) twice with two different programmed challenges that only differ in the element \( c^* \) (\( R \) and \( e \) stay the same). Rewind takes an algorithm \( \text{Gen} \) that generates \( (\text{prm}, \text{vk}, sk, \text{hlp}, T) \), where \( T \) is the table used by \( H \). Rewind simulates \( H \) for \( A \) using \( H_r \) as described by the code. Rewind returns the success status of the rewinding process, along with the two responses of the two executions \( (z_1, z_2) \), plus the honest response \( (z^*) \) and the
We now justify Equation (1.29). We consider the event that the flags both exactly \( \Pr \). Notice that the marginal probability that \( A \) is true and the marginal probability that \( B \) is true are both exactly \( \Pr \). We partition the random tape for \( G_{\text{ID},\ell}^\text{imp}(\mathcal{A}) \) into two parts: the random \[ \Pr\{\text{Game Rewind}^1(\text{Gen, } \mathcal{A}) | \text{Rewind}^2(\text{Gen, } \mathcal{A})\} \leq \text{Adv}_{\text{ID},\ell}^\text{imp}(\mathcal{A})^2 - \frac{1}{p}. \]

We now justify Equation (1.29). We consider the event that the flags \( A, B, C \) are all set to true. Notice that the marginal probability that \( A \) is true and the marginal probability that \( B \) is true are both exactly \( \Pr \). We partition the random tape for \( G_{\text{ID},\ell}^\text{imp}(\mathcal{A}) \) into two parts: the random
### Adversary $\mathcal{A}_{cdh}(G, v, h)$

- $(G, G_T, g, e, p) \leftarrow \mathcal{G}$
- $(t, z_1, z_2, z^*, sk^*) \leftarrow $ Rewind$^2(G_{cdh}, \mathcal{A})$
- $(pk_1^*, \sigma_1^*, z^{(1)}) \leftarrow z_1$
- $(pk_2^*, \sigma_2^*, z^{(2)}) \leftarrow z_2$
- $(pk^*, \sigma^*, z^*) \leftarrow z^*$
- $\hat{\sigma} \leftarrow ((\sigma_1^*)^{c_1}/(\sigma_2^*)^{c_2}) \cdot \sigma^{c_2 - c_1}$
- $\omega = \sum_{j=1}^{m} \gamma_j (z^{(1)}_j - z^{(2)}_j - x_j^* (c_1^* - c_2^*))$
- $\delta' \leftarrow (\hat{\sigma})^{1/\omega}$
- Return $\delta'$

### Gen$\mathcal{G}_{cdh}()$

- $(G, G_T, g, e, p) \leftarrow \mathcal{G}$
- For $j \in [m]$ do
  - $\gamma_j \leftarrow \mathbb{Z}_p$; $g_j \leftarrow h^{\gamma_j}$
  - $\text{prm} = (g_0, \ldots, g_{m-1}, g)$; $\text{vk} \leftarrow v$
- For $i \in [m]$ do
  - $sk[i] \leftarrow [p]^m$; $pk[i] \leftarrow \prod_{j=0}^{m-1} g_j^{sk[i][j]}$
  - $\beta_i \leftarrow [p]$; $\sigma[i] \leftarrow \text{vk}^{\beta_i}$
  - $T[i, G] \leftarrow g^{\beta_i}/pk[i]$
- $\text{hlp} \leftarrow (pk, \sigma)$
- Return $(\text{prm}, \text{vk}, \text{sk}, \text{hlp}, T)$

### Adversary $\mathcal{A}_{dl}(G, X)$

- $(t, z_1, z_2, z^*, sk^*) \leftarrow $ Rewind$^2(G_{dl}, \mathcal{A})$
- $(pk_1^*, \sigma_1^*, z^{(1)}) \leftarrow z_1$
- $(pk_2^*, \sigma_2^*, z^{(2)}) \leftarrow z_2$
- $(pk^*, \sigma^*, z^*) \leftarrow z^*$
- For $j = 1, \ldots, m$ do
  - $sk^*[j] \leftarrow (z^{(1)}[j] - z^{(2)}[j])/(c_1^* - c_2^*)$
  - $x \leftarrow (\sum_{i=m}^{m-1} x_i (sk^*[i] - sk^*[i]))/(sk^*[p] - sk^*[p])$
- Return $x$

### Gen$\mathcal{G}_{dl}()$

- $\rho \leftarrow [m]$; $g_\rho \leftarrow X$
- For $j \in [m] - \{\rho\}$ do
  - $x_j \leftarrow \mathbb{Z}_p$; $g_j \leftarrow g^{x_j}$
- $\text{prm} = (g_0, \ldots, g_{m-1}, g)$
- $(pk, sk, hlp) \leftarrow \text{ADW.KeyGen}($prm$)$
- Return $(\text{prm}, pk, sk, hlp, \bot)$

---

**Figure 1.14:** **Left:** Adversary $\mathcal{A}_{cdh}$. **Right:** Adversary $\mathcal{A}_{dl}$.

Tape that is used up to right before $\mathcal{A}.\text{Rsp}$ is run, and the rest of the tape that is used after $\mathcal{A}.\text{Rsp}$ starts its execution. Let $T$ be a random variable denoting the first part of the random tape. For any value of $T$, say $t$, we let $G(t)$ be the game $G_{\text{ID}, t}^{\text{imp}}(\mathcal{A})$ with the first part of random tape fixed to $t$. 

---

54
We have that

\[
\Pr \left[ \text{Rewind}^1(\text{Gen}, \mathcal{A}) \right] = \Pr [A \land B \land C] \\
= E_T [\Pr [A \land B \land C \mid T]] \\
\geq E_T [\Pr [A \land B \mid T] - \Pr [\neg C \mid T]] \\
= E_T [\Pr [\mathbf{G}(T)]]^2 - \frac{1}{p} \\
\geq \Pr \left[ \mathbf{G}_{\text{id}, \ell}^{\text{imp}}(\mathcal{A}) \right]^2 - \frac{1}{p} ,
\]

where at the last step we used Jensen’s inequality and the convexity of squaring. This justifies Equation (1.29). Second, we argue that

\[
\Pr[\text{Rewind}^1(\text{Gen}, \mathcal{A})] - \Pr[\text{Rewind}^2(\text{Gen}, \mathcal{A})] \leq \Pr[\text{Rewind}^1(\text{Gen}, \mathcal{A}) \text{ sets bad}] = \frac{q}{2r} . \tag{1.30}
\]

This is because the size of the table \( C \) is upper-bounded by the number of queries that \( \mathcal{A}.\text{Setup} \) and \( \mathcal{A}.\text{Com} \) makes to \( H \) and \( \text{Derive} \), which is \( q \). Next, we attempt to bound \( \Pr[\text{Rewind}^2(\text{Gen}, \mathcal{A})] \).

We define the following events in the game \( \text{Rewind}^2(\text{Gen}, \mathcal{A}) \).

\[
E : \ A \land B \land C \land \left( \frac{(pk_1^*)^{(c_1)}}{(pk_2^*)^{(c_2)}} = (pk^*)^{c_1 - c_2} \right), \\
\overline{E} : \ A \land B \land C \land \left( \frac{(pk_1^*)^{(c_1)}}{(pk_2^*)^{(c_2)}} \neq (pk^*)^{c_1 - c_2} \right) .
\]

Notice that per construction of the events,

\[
\Pr[\text{Rewind}^2(\text{Gen}, \mathcal{A})] = \Pr[E] + \Pr[\overline{E}] . \tag{1.31}
\]
Consider $A_{cdh}$ (Fig. 1.14) and $A_{dl}$ (Fig. 1.14), which attempts to break CDH and DL problems, respectively, using Rewind$^2$. We will show the following (in)equalities

$$\Pr[E] = \text{Adv}^{cdh}_G(A_{cdh}),$$  

(1.32)

and

$$\Pr[E] \leq m \cdot \text{Adv}^{dl}_G(A_{dl}) + \text{Adv}^{pskp}_{p,m,k,\tau}(\ell + \frac{k}{m}).$$  

(1.33)

This part of the analysis follows from [6, Appendix B.5] and we restate their derivation here. Assume $E$ or $\overline{E}$, since the signatures verifies, for $w = \prod_{i=0}^{\tau} H(pp[i], G)^{e_i}$, we have

$$\sigma^* = (pk^*w)^s, \quad \sigma_1^* = (pk_1^*w)^s, \quad \sigma_2^* = (pk_2^*w)^s.$$

If $\overline{E}$ happens, the following two values are distinct

$$\frac{(\sigma_1^*)^{c_1}}{(\sigma_2^*)^{c_2}} = \left(\frac{(pk_1^*)^{c_1}}{(pk_2^*)^{c_2}}\right)^s, \quad (\sigma^*)^{c_1 - c_2} = \left(\frac{w^{c_1 - c_2}(pk)^{c_1 - c_2}}{g_0}\right)^s.$$

Hence, the value $\hat{\sigma}$ computed by $A_{cdh}$ is

$$\hat{\sigma} = \left(\frac{(pk_1^*)^{c_1}}{(pk_2^*)^{c_2}} \cdot \frac{1}{(pk^*)^{c_1 - c_2}}\right)^s = (g_0^0)^s.$$

Therefore, $A_{cdh}$ can compute $g^s$ and solve the CDH problem that it was given. This concludes the proof for Equation (1.32). If $E$ happens, then we claim that $sk^* = \hat{sk}$ with probability at most $\text{Adv}^{pskp}_{p,k,\tau}(\ell + k/m)$. This is true per definition of $\text{Adv}^{pskp}_{p,k,\tau}(\ell + k/m)$. Notice that if $sk^* \neq \hat{sk}$, then with probability $1/m$, $A_{dl}$ can solved the DL problem that it embedded into the parameters. This is because $A_{dl}$ has two representation of $pk^*$ in the basis $g_0, \ldots, g_{m-1}$, namely $sk^*$ and $\hat{sk}$. This concludes the proof of Equation (1.33). Notice that Equations (1.29), (1.31), (1.30), (1.32), and (1.33) together implies the theorem. Finally, notice that $A_{cdh}$ and $A_{dl}$ has roughly the running
time of Rewind$^2$ and ADW.KeyGen, which is about $t_1 + t_2 + 2t_3 + t_4$.

1.6 Proofs of Lemmas 1.3.2, 1.3.3, and 1.3.4

We need the following version of Stirling’s approximation of $n!$.

**Lemma 1.6.1**\[77\] For any $n \in \mathbb{Z}^+$,

$$\sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{\frac{1}{12n+1}} \leq n! \leq \sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{\frac{1}{12n}}.$$

We first prove Lemma 1.3.2.

**Proof of Lemma 1.3.2** We first show the lower bound Equation (1.6). Notice that by definition of $H_q(r/k)$,

$$q^{kH_q(r/k)} = (q - 1)^r(r/k)^{-r}(1 - r/k)^{r-k}.$$

Hence, by Lemma 1.6.1,

$$B_{q,k}(r) = \sum_{i=0}^{r} (q - 1)^i \binom{k}{i}$$

$$\geq (q - 1)^r \frac{k!}{r!(k-r)!}$$

$$\geq (q - 1)^r \frac{\sqrt{2\pi k(k/e)^k e^{1/12k+1}}}{\sqrt{2\pi r(r/e)^r e^{1/12r+1}} \sqrt{2\pi (k-r)(k-r)/e^{1/12(k-r)}}}$$

$$= q^{kH_q(r/k)} \frac{\sqrt{k e^{1/12k+1}}}{\sqrt{2\pi r(k-r)e^{1/12(k-r)}}}$$

$$= q^{kH_q(r/k) - e(k,r)}.$$

57
Now, we assume that $r \leq k - k/q$ and derive the upper bound, Equation (1.7).

\[
\frac{B_{q,k}(r)}{q^{kH_q(r/k)}} = \frac{\sum_{i=0}^{r-1}(q-1)^i \binom{k}{i}}{(q-1)^r(r/k)^{-r}(1-r/k)^{r-k}}
\]

\[
= \sum_{i=0}^{r} \binom{k}{i} (q-1)^i (r/k)^{-r}(1-r/k)^{k-r}
\]

\[
= \sum_{i=0}^{r} \binom{k}{i} (q-1)^i (1-r/k)^k \left( \frac{r/k}{(q-1)(1-r/k)} \right)^r
\]

\[
\leq \sum_{i=0}^{r} \binom{k}{i} (q-1)^i (1-r/k)^k \left( \frac{r/k}{(q-1)(1-r/k)} \right)^i
\]

\[
= \sum_{i=0}^{r} \binom{k}{i} (r/k)^i (1-r/k)^{k-i}
\]

\[
\leq \sum_{i=0}^{r} \binom{k}{i} (r/k)^i (1-r/k)^{k-i}
\]

\[
= 1,
\]

where the first inequality is by the fact that $r/k \leq (q-1)(1-r/k)$ if $r \leq k - k/q$. \ Lemma 1.3.3

follows from Lemma 1.3.2 \ Proof of Lemma 1.3.3 \ Per definition of $rd_q(k,N)$, it suffices to show that

\[
B_{q,k}(r) \leq N,
\]

for $r = \left\lfloor H_q^{-1}(\log_q(N)/k) \right\rfloor$. Per definition of $H_q^{-1}$, $r \leq (1 - 1/q) \cdot k$. Hence, we can apply Equation (1.7) and obtain

\[
B_{q,k}(r) \leq q^{kH_q(r/k)} \leq q^{kH_q(H_q^{-1}(\log_q(N)/k))} = N. \]

Lastly, we prove Lemma 1.3.4.
Proof of Lemma 1.3.4: We first show the lower bound that

\[ \min(x, 1 - \frac{1}{q}) - \frac{1}{\log_2(q)} \leq H_q^{-1}(x). \]  \tag{1.34} \]

Note that this is trivially true if the left-hand side of Equation (1.34) is negative. Hence, we suppose that the left-hand side of Equation (1.34) is non-negative. As noted before, \( H_q \) is increasing in the domain \([0, 1 - 1/q]\). Additionally, note that \( \min(x, 1 - 1/q) - 1/\log_2(q) \leq 1 - 1/q \). Hence, it suffices to show

\[ H_q \left( \min(x, 1 - \frac{1}{q}) - \frac{1}{\log_2(q)} \right) \leq x. \]  \tag{1.35} \]

We consider two cases. Case 1, \( x \leq (1 - 1/q) \). Case 2, \( (1 - 1/q) \leq x \leq 1 \). We claim that both cases follow from the equation below, which holds for \( x \in [\log_2(q), 1] \).

\[ H_q \left( x - \frac{1}{\log_2(q)} \right) \leq x. \]  \tag{1.36} \]

Case 1 is directly implied by Equation (1.36). For case 2, note that the left-hand side of Equation (1.34) always evaluate to \( 1 - 1/q - 1/\log_2(q) \). Hence, by Equation (1.36), \( H_q(1 - 1/q - 1/\log_2(q)) \leq 1 - 1/q \leq x \). Finally, we justify Equation (1.36). Recall that \( H_q(x) = \)
\( H_2(x) / \log_2(q) + x \log_q(q - 1) \). We compute

\[
H_q(x - \frac{1}{\log_2(q)}) = \frac{H_2(x - \frac{1}{\log_2(q)})}{\log_2(q)} + \left( x - \frac{1}{\log_2(q)} \right) \log_q(q - 1)
\]

\[
\leq \frac{1}{\log_2(q)} + x \log_q(q - \frac{1}{q}) - \frac{\log_q(q - 1)}{\log_2(q)}
\]

\[
= \frac{1}{\log_2(q)} + x - x \log_q(q - 1) - \frac{\log_q(q - 1)}{\log_2(q)}
\]

\[
= x + \frac{1}{\log_2(q)} (1 - \log_q(q - 1))
\]

\[
\leq x + \frac{1}{\log_2(q)}(1 - \log_q(q))
\]

\[
= x.
\]

Next, we show the upper bound that

\[
H_q^{-1}(x) \leq x \left( 1 - \frac{1}{q} \right).
\] (1.37)

Similar to the lower bound we just obtained, we note that it suffices to show \( H_q(x \left( 1 - \frac{1}{q} \right)) \geq x \).

Let us define, for \( x \in [0, 1] \):

\[
f(x) = \frac{x}{q} \log_q(x) - x \log_q(1 - \frac{x}{q}) \leq (1 - x + \frac{x}{q}) \log_q\left( 1 - x + \frac{x}{q} \right).
\]
We will show that $H_q(1(1-1/q)) = x + f(x)$. The derivation is as follows.

\[
H_q(x(1-1/q))
\]

\[
= x(1-1/q) \log_q (q-1) - x(1-1/q) \log_q (x(1-1/q))
\]

\[
- (1-x+x/q) \log_q (1-x+x/q)
\]

\[
\geq x \log_q (q-1) - x/q \log_q (q-1)
\]

\[
- x(1-1/q)(\log_q (x) + \log_q (1-1/q))
\]

\[
- (1-x+x/q) \log_q (1-x+x/q)
\]

\[
= x \log_q (q-1) - x/q \log_q (q-1)
\]

\[
- x \log_q (x) - x \log_q (1-1/q) + x/q \log_q (x) + x/q \log_q (1-1/q)
\]

\[
- (1-x+x/q) \log_q (1-x+x/q)
\]

\[
= x \left( \log_q (q-1) + \log_q (q/(q-1)) \right) - x \log_q (x)
\]

\[
- x/q \left( \log_q (q-1) + \log_q (1/x) + \log_q (q/(q-1)) \right)
\]

\[
- (1-x+x/q) \log_q (1-x+x/q)
\]

\[
= x + x/q \log_q (x/q) - x \log_q (x) - (1-x+x/q) \log_q (1-x+x/q)
\]

\[
= x + f(x)
\]

Lastly, we show that $f(x) \geq 0$ for any $x \in [0,1]$. First, we check that $f(0) = f(1) = 0$. Next, check that the second derivative of $f$,

\[
f''(x) = \frac{q-1}{x(qx-q-x)} \leq 0,
\]

is at most 0 for any $x \in [0,1]$. We omit the details of the derivative computation here. Hence, $f$ is
concave over the domain $[0, 1]$, with $f(0) = f(1) = 0$. Thence, $f(x) \geq 0$ for all $x \in [0, 1]$. 

\section{1.7 A Rejection Sampling Lemma}

We prove the following lemma, which allows one to efficiently sample from $[k]^\tau$, for appropriately constrained integers $k, \tau$, via rejection sampling.

\textbf{Lemma 1.7.1} \textit{Let $\tau, k, c$ be positive integers. Suppose $k \geq 2(c + 1) \cdot \tau^2$. Let $x_1, \ldots, x_t \leftarrow [k]$ be i.i.d samples, where}

$$
t = \tau + \left\lceil \frac{c\tau}{\log_2(k) - \log((c + 1)\tau^2)} \right\rceil \leq \tau + c\tau.
$$

\textit{Let $S = \{x_1, \ldots, x_t\}$. Then}

$$
\Pr[|S| < \tau] \leq 2^{-c \cdot \tau}.
$$

\textbf{Proof:} Let $\delta = t - \tau$. Since $k \geq 2(c + 1) \cdot \tau^2$, we have $\log_2(k) - \log((c + 1)\tau^2) \geq 1$. Hence, $\delta \leq c \cdot \tau$. Define $S_i = \{x_1, \ldots, x_i\}$. Hence, $S_0 = \emptyset$ and $S_t = S$. Suppose that $|S| < \tau$, then there exists at least $\delta$ positions $i$ such that $x_i \in S_{i-1}$. Since $x_i$ is a independent uniform sample from $[k]$, the probability that $x_i \in S_{i-1}$ is $|S_{i-1}|/k$, which is at most $\tau/k$. Hence,

$$
\Pr[|S| < \tau] \leq \left( \frac{\tau + \delta}{\delta} \right) \left( \frac{\tau}{k} \right)^\delta
\leq \left( \frac{(c + 1)\tau}{\delta} \right) \left( \frac{\tau}{k} \right)^\delta
\leq \left( \frac{(c + 1)\tau^2}{k} \right)^\delta.
$$

Hence,

$$
\log_2(\Pr[|S| < \tau]) \leq \delta \log_2\left( \frac{(c + 1)\tau^2}{k} \right) \leq -c \cdot \tau.
$$
1.8 Acknowledgements

We thank the anonymous CCS 2018 reviewers for their helpful comments. We thank Yevgeniy Dodis and Daniel Wichs for helpful discussions.

Bellare was supported in part by NSF grants CNS-1526801 and CNS-1717640, ERC Project ERCC FP7/615074 and a gift from Microsoft corporation. Dai was supported in part by a Powell Fellowship and grants of Bellare.

Chapter 1, in full, is a reprint of the material as it appears in ACM Conference on Computer and Communications Security, 2017. Bellare, Mihir; Dai, Wei. The dissertation author was the primary investigator and author of this paper.
Chapter 2

Tight and Non-rewinding Proofs for Schnorr Identification and Signatures

2.1 Introduction

It would not be an exaggeration to say that Schnorr identification and signatures [79] are amongst the best-known and most influential schemes in cryptography. With regard to practical impact, consider that Ed25519, a Schnorr-derived signature scheme over twisted Edwards curves [20], is used, according to IANIX [55], in over 200 different things. (OpenSSL, OpenSSH and GnuPG to name a tiny fraction.) Meanwhile the algebraic structure of the Schnorr schemes has resulted in their being the basis for many advanced primitives including multi- [66, 14, 8, 63], ring- [2, 53] and threshold- [84, 59] signatures.

Proving security of these schemes has accordingly attracted much work. Yet, all known standard-model proofs [76, 1] [58] exhibit a gap: the proven bound on adversary advantage (success probability) is much inferior to (larger than) the one that cryptanalysis says is “true.” (The former is roughly the square-root of the latter. Accordingly we will refer to this as the square-root gap.)
The square-root gap is well known and acknowledged in the literature. Filling this long-standing and notorious gap between theory and practice is the subject of this paper. We start with some background.

**Schnorr schemes.** Let $\mathbb{G}$ be a group of prime order $p$, and $g \in \mathbb{G}$ a generator of $\mathbb{G}$. We let $\text{ID} = \text{SchID}[\mathbb{G}, g]$ denote the Schnorr identification scheme \[79\] (shown in Figure 2.3). The security goal for it is IMP-PA (impersonation under passive attack \[44\]). The Schnorr signature scheme $\text{DS} = \text{SchSig}[\mathbb{G}, g]$ \[79\] is derived from ID via the Fiat-Shamir transform \[45\] (also shown in Figure 2.3). The security goal for it is UF (unforgeability under chosen-message attack \[50\]) in the ROM (random oracle model \[18\]).

Recall that, $\mathbb{G}, g$ being public, the DL problem is for an adversary, given $X = g^x$, to recover $x$. Since we will introduce variants, we may, for emphasis, refer to DL itself as the “basic” version of the discrete-logarithm problem. Existing standard-model proofs for both ID and DS \[76, 1, 58\] are based on the assumed hardness of DL. The heart of the proof for DS, and the cause of the square-root gap, is the rewinding reduction in the proof for ID. This makes ID the first and most basic concern.

**The situation with ID.** The simplest proof of IMP-PA for $\text{ID} = \text{SchID}[\mathbb{G}, g]$ uses the Reset Lemma of \[15\]. It shows that, roughly:

\[
\epsilon^{\text{imp-pa}}(t) \leq \sqrt{\epsilon^{\text{dl}}(t)} + \frac{1}{p},
\]

where $\epsilon^{\text{imp-pa}}(t)$ is the probability of breaking IMP-PA security of ID in time $t$ and $\epsilon^{\text{dl}}(t)$ is the probability of breaking DL in time $t$. To draw quantitative conclusions about $\epsilon^{\text{imp-pa}}(t)$ as required in practice, however, we now also need to estimate $\epsilon^{\text{dl}}(t)$. The accepted way to do this is via the Generic Group Model (GGM) bound \[81\], believed to be accurate for elliptic curve groups. It
says that
\[ \varepsilon_{\text{dl}}(t) \approx \frac{t^2}{p}. \] (2.2)

Putting together the two equations above, we get, roughly:
\[ \varepsilon_{\text{imp-pa}}(t) \leq \frac{t}{\sqrt{p}}. \] (2.3)

There is, however, no known attack matching the bound of Eq. (2.3). Indeed, the best known time \( t \) attack on ID is via discrete-log computation and thus has the considerably lower success probability of \( t^2/p \). For example if \( p \approx 2^{256} \) the best known attack against ID gives a time \( t = 2^{80} \) attacker a success probability of \( t^2/p = 2^{-96} \), but Eq. (2.3) only rules out a success probability of \( t/\sqrt{p} = 2^{-48} \). The proof is thus under-estimating security by a fairly large margin.

Accordingly in practice the proof is largely viewed as a qualitative rather than quantitative guarantee, group sizes being chosen in ad hoc ways. Improving the reduction of Eq. (2.1) to bring the theory more in line with the indications of cryptanalysis has been a long-standing open question.

** TIERS AND KNOBS.** Before continuing with how we address this question, we draw attention to the two-tiered framework of a security proof for a scheme \( S \) (above, \( S = \text{ID} \)) based on the assumed hardness of some problem \( P \) (above, \( P=\text{DL} \)). The first tier is the reduction from \( P \). It is represented above by Eq. (2.1). The second tier is the estimate of the security of \( P \) itself, made (usually) in an idealized model such as the GGM [81] or AGM (Algebraic Group Model) [47]. It is represented above by Eq. (2.2). Both tiers are necessary to draw quantitative conclusions. This two-tier structure is an accepted one for security proofs, and widely, even if not always explicitly, used.

In this structure, we have the flexibility of choice of \( P \), making this a “knob” that we can tune. Creative and new choices of \( P \) have historically been made, and been very valuable.
in cryptography, yielding proofs for existing schemes and then going on to be useful beyond. Historically, a classical example of such a (at the time, new) $P$ is the Diffie-Hellman problem; the schemes $S$ whose proof this allows include the Diffie-Hellman secret-key exchange [39] and the El Gamal public-key encryption scheme [43]. An example $P$ closer to our work is the One-More Discrete Logarithm (OMDL) problem [13], which has by now been used to prove many schemes $S$ [15, 37, 75, 46, 40]. But this knob-tuning approach is perhaps most visible in the area of bilinear maps, where many choices of problem $P$ have been made, justified in the GGM, and then used to prove security of many schemes $S$. In the same tradition, we ask, how can we tune the knob to fill the square-root gap? Our answer is a choice of $P$ we call MBDL.

**MBDL.** Our Multi-Base Discrete Logarithm (MBDL) problem is a variant of the One-More Discrete Logarithm (OMDL) problem of [13]. Continue to fix a cyclic group $G$ and generator $g$ of $G$. In MBDL, the adversary is given a challenge $Y \in G$, a list $X_1, X_2, \ldots, X_n \in G^*$ of generators of $G$, and access to an oracle $DLO$ that, on input $i, W$, returns $DL_{G,X_i}(W)$, the discrete logarithm of $W$, *not in base $g$, but in base $X_i$*. To win it must find $DL_{G,g}(Y)$, the discrete logarithm of the challenge $Y$ to base $g$, while making at most one call to $DLO$ overall, meaning it is allowed to take the discrete log of at most one group element. (But this element, and the base $X_i$, can be chosen as it wishes.) The number of bases $n$ is a parameter of the problem, so that one can refer to the $n$-MBDL problem or assumption. (Our results will rely only on 1-MBDL, but we keep the definition general for possible future applications.) The restriction to at most one $DLO$ call is necessary, for if even two are allowed, $DL_{G,g}(Y)$ can be obtained as $DLO(1,Y) \cdot DLO(1,g)^{-1} \mod p$ where $p = |G|$.

**Core Results.** We suggest that the square-root gap of Eq. (2.1) is a manifestation of an unformalized strength of the discrete logarithm problem. We show that this strength is captured by the MBDL problem. We do this by giving a proof of IMP-PA security of the Schnorr identification scheme $ID = SchlID[G,g]$ with a *tight* reduction from 1-MBDL: letting $\epsilon^{1\text{-mbdl}}(t)$
Table 2.1: Speedups yielded by our results for the Schnorr identification scheme ID = SchID[G, g] (top) and signature scheme DS = SchSig[G, g] (bottom). The target for the first is that IMP-PA adversaries with running time \( t \) should have advantage at most \( \epsilon \). We show the log of the group size \( p_i \) required for this under prior results \( (i = 1) \), and our results \( (i = 2) \). Assuming exponentiation in \( G \) is cubic-time, we then show the speedup ratio of scheme algorithms. The target for the second is that UF adversaries with running time \( t \), making \( q_h \) queries to \( H \), should have advantage at most \( \epsilon \), and the table entries are analogous.

### Schnorr Identification

<table>
<thead>
<tr>
<th>( t )</th>
<th>( \epsilon )</th>
<th>( \log(p_1) )</th>
<th>( \log(p_2) )</th>
<th>Speedup ( s = (\log(p_1) / \log(p_2))^3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 2^{80} )</td>
<td>( 2^{-48} )</td>
<td>256</td>
<td>208</td>
<td>1.9</td>
</tr>
<tr>
<td>( 2^{64} )</td>
<td>( 2^{-64} )</td>
<td>256</td>
<td>192</td>
<td>2.4</td>
</tr>
<tr>
<td>( 2^{100} )</td>
<td>( 2^{-156} )</td>
<td>512</td>
<td>356</td>
<td>3</td>
</tr>
</tbody>
</table>

### Schnorr Signatures

<table>
<thead>
<tr>
<th>( t )</th>
<th>( q_h )</th>
<th>( \epsilon )</th>
<th>( \log(p_1) )</th>
<th>( \log(p_2) )</th>
<th>Speedup ( s = (\log(p_1) / \log(p_2))^3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 2^{80} )</td>
<td>( 2^{60} )</td>
<td>( 2^{-48} )</td>
<td>316</td>
<td>268</td>
<td>1.6</td>
</tr>
<tr>
<td>( 2^{64} )</td>
<td>( 2^{50} )</td>
<td>( 2^{-64} )</td>
<td>306</td>
<td>242</td>
<td>2.0</td>
</tr>
<tr>
<td>( 2^{100} )</td>
<td>( 2^{80} )</td>
<td>( 2^{-156} )</td>
<td>592</td>
<td>436</td>
<td>2.5</td>
</tr>
</tbody>
</table>

be the probability of breaking the 1-MBDL problem in time \( t \), Theorem \( 2.4.1 \) says that, roughly:

\[
\epsilon^{\text{imp-pa}}(t) \leq \epsilon^{\text{1-mbdl}}(t) + \frac{1}{p}, \tag{2.4}
\]

Eq. (2.4) does not suffer from the square-root gap of Eq. (2.1). Progress. But this is in the first of the two tiers discussed above. Turning to the second, we ask, how hard is MBDL? Theorem \( 2.5.1 \) shows that, in the GGM, roughly:

\[
\epsilon^{\text{1-mbdl}}(t) \approx \frac{t^2}{p}. \tag{2.5}
\]

That is, 1-MBDL problem has essentially the same GGM quantitative hardness as DL. Putting
Eqs. (2.4) and (2.5) together, we get (roughly) the following improvement over Eq. (2.3):

$$\varepsilon_{\text{imp-pa}}(t) \leq \frac{t^2}{p}.$$  \hfill (2.6)

This bound is tight in the sense that it matches the indications of cryptanalysis.

A direct indication of the practical value of this improvement is that, for a given target level of provable security, we increase efficiency. Thus suppose that, for some chosen values of $\varepsilon, t$, we want to pick the group $\mathbb{G}$ to ensure $\varepsilon_{\text{imp-pa}}(t) \leq \varepsilon$. Eq. (2.6) allows us to use smaller groups than Eq. (2.3). Since scheme algorithms have running time cubic in the log of $p = |\mathbb{G}|$, this results in a performance improvement. Figure 2.1 says that this improvement can range from 1.9x to 3x.

**WHAT HAS BEEN GAINED?** A natural question is that our results rely on a new assumption (MBDL), so what has been gained? Indeed, MBDL, as with any new assumption, should be treated with caution. However, it seems that improving Eq. (2.1) to something like Eq. (2.4) under the basic DL assumption is out of reach and likely not possible, and thus that, as indicated above, the apparent strength of the Schnorr schemes indicated by cryptanalysis is arising from stronger hardness properties of the discrete log problem not captured in the basic version. We are trying to understand and formalize this hardness via new problems that tightly imply security of the Schnorr primitives.

Of course it would not be hard to introduce some problem which allows this. But we believe MBDL, and our results, are “interesting” in this regard, for the following reasons. First, MBDL is not a trivial reformulation of the IMP-PA security of lD, meaning we are not just assuming the square-root problem out of existence. Second, and an indication of the first, is that the proof of the IMP-PA security of lD from MBDL (see “Reduction approach” below) is correspondingly not trivial. Third, the use of MBDL is not confined to Schnorr identification; as we also discuss below under “MBDL as a hub,” it already has many further applications and uses,
and we imagine even more will arise in the future.

**Reduction Approach.** The proof of Eq. (2.1) uses a rewinding argument that exploits the special soundness property of the Schnorr identification scheme, namely that from two compatible transcripts —this means they are accepting and have the same commitment but different challenges— one can extract the secret key. To find the discrete log, in base $g$, of a given challenge $Y$, the discrete log adversary $\mathcal{B}$ plants the challenge as the public key $X$ and performs two, related runs of the given IMP-PA adversary, hoping to get two compatible transcripts, in which case it can extract the secret key and solve its DL instance. The Reset Lemma [15] says it is successful with probability roughly the square of the IMP-PA advantage of $\mathcal{A}$, leading to the square-root in Eq. (2.1).

Recall that our 1-mbdl adversary $\mathcal{B}$ gets input a challenge $Y$ whose discrete logarithm *in the usual base $g$ it must find*, just like a DL adversary. To get Eq. (2.4) we must avoid rewinding. The question is how and why the ability to take one discrete logarithm in some random base $X_1$ helps to do this and get a tight reduction. Our reduction deviates from prior ones by *not* setting $Y$ to the public key. Instead, it sets $X_1$ to the public key. Then, it performs a single execution of the given IMP-PA adversary $\mathcal{A}$, “planting” $Y$ in the communication in such a way that success of $\mathcal{A}$ in impersonating the prover yields $\text{DL}_{g}(Y)$. This planting step makes one call to $\text{DLO}(1,\cdot)$, meaning asks for a discrete logarithm in base $X_1$ of some $W$ that depends on the execution. The full proof is in Section 2.4.

**MBDL as a Hub.** Having identified MBDL, we find that its applicability extends well beyond what is discussed above, making it a hub. Here we briefly discuss further results from MBDL.

The Schnorr signature scheme $\text{DS} = \text{SchSig}[G,g]$ has a proof of UF-security in the ROM under the basic DL assumption [76,73,1,58]. The bound —recalled in Eq. (2.15)— continues to exhibit the square-root gap. Theorem 2.4.3 gives a square-root avoiding reduction from 1-MBDL to fill this gap. Figure 2.1 shows resulting speedup factors of 1.6x to 2.5x for Schnorr signatures.
Security above refers to the single-user setting. Our results extend to tightly reduce the multi-user IMP-PA security of SchID[$G, g$] to 1-MBDL, and analogously for signatures. This can be shown directly, but is also a consequence of general results of [58].

The situation for the Okamoto identification and signature schemes [74] is analogous to that for Schnorr, meaning the reductions in the current security proofs, from DL, use rewinding and has the square-root loss. In Section 2.6 we give results for Okamoto that are analogous to our results for Schnorr, meaning reductions from 1-MBDL that avoid the square root.

There’s more. In a follow-up work, we also give reductions from MBDL that improve security of the following: (1) Bellare-Neven multi-signatures [14] (2) Abe, Ohkubo, Suzuki 1-out-of-n (ring/group) signatures [2] and (3) Schnorr-based threshold signatures [84].

**RELATED WORK.** One prior approach to resolving the square-root gap has been to use only an idealized model like the GGM [81] or AGM [47]. Thus, Shoup [81] directly showed that $\epsilon^{imp-pa}(t) \leq t^2 / p$ in the GGM. Fuchsbauer, Plouviez and Seurin [48] give, in the AGM, a tight reduction from DL to the UF security of DS = SchSig[$G, g$]. These results correspond to a setting of the knob, in the above-discussed two-tier framework, that is maximal: P is the target scheme itself (here Schnorr identification or signatures), so that the first tier is trivial and the second tier directly proves the scheme secure in the idealized model.

But it is well understood that idealized models have limitations. Proofs in the GGM assume the adversary does not exploit the representation of group elements. In the AGM, it is assumed that, whenever an adversary provides a group element Z, it is possible to extract its representation as a product of known powers of prior group elements. This is analogous to a “knowledge of exponent assumption” [35, 51, 16]. However, even in a typical elliptic curve group, an adversary can quite easily create group elements without “knowing” such a representation. The maximal setting of knob (working purely in an idealized model) means the security guarantee on the scheme is fully subject to the limitations of the idealized model.

With MBDL, we, instead make a non-trivial, moderate setting of the knob. Our tight
reductions from MBDL, such as Eq. (2.4), are in the standard model, and make no GGM or AGM-like assumptions on adversaries. It is of course true that we justify MBDL in the GGM (Theorem [2.5.1]), but we are limiting the use of the idealized model to show security for a purely number-theoretic problem, namely MBDL. The first direct benefit is better security guarantees for the schemes. The second is that MBDL is a hub. As discussed above, we can prove security of many schemes from it, which reduces work compared to proving them all from scratch in idealized models, and also increases understanding by identifying a problem that is at the core of many things.

Another prior approach to improving reduction tightness has been to change metrics, measuring tightness, not via success probability and running time taken individually, but via their ratio [58]. This however does not translate to actual, numeric improvements. To discuss this further, let IMP-KOA denote impersonation under key-only attack. (That is, IMP-PA for adversaries making zero queries to their transcript oracle.) Kiltz, Masny and Pan (KMP) [58] define a problem they call 1-IDLOG that is a restatement of (“precisely models,” in their language) the IMP-KOA security of \( \text{ID} = \text{SchID}[G,g] \). Due to the zero knowledge of \( \text{ID} \), its IMP-PA security reduces tightly to its IMP-KOA security and thus to 1-IDLOG. Now, KMP [58] give a reduction of 1-IDLOG to DL that is ratio-tight, meaning preserves ratios of advantage to running time. This, however, uses rewinding, and is not tight in our sense, incurring the usual square-root loss when one considers running time and advantage separately. In particular the results of KMP do not seem to allow group sizes any smaller than allowed by the classical Eq. (2.1). Our reductions, in contrast, are tight for advantage and time taken individually, and across the full range for these values, and numerical estimates (Figure 2.1) show clear improvements over what one gets from Eq. (2.1). Also our results establish 1-IDLOG tightly (not merely ratio-tightly) under 1-MBDL. We discuss ratio-tightness further in Section 2.7.

**DISCUSSION.** Measuring quality of a reduction in terms of bit security effectively only reflects the resources required to attain an advantage close to 1. Under this metric, whether one
starts from Eq. (2.1) or Eq. (2.4), one concludes that $ID = \text{SchlD}[G, g]$ has $\log_2(|G|)/2$-bits of security. This reflects bit security being a coarse metric. The improvement offered by Eq. (2.4) over Eq. (2.1) becomes visible when one considers the full curve of advantage as a function of runtime, and is visible in Figure 2.1.

While new assumptions (like MBDL) should of course be treated with caution, cryptographic research has a history of progress through introducing them. For example, significant advances were obtained by moving from the CDH assumption to the stronger DDH one [67, 33]. Pairing-based cryptography has seen a host of assumptions that have had many further applications, including the bilinear Diffie-Hellman (BDH) assumption of [26] and the DLIN assumption of [24]. The RSA $\Phi$-Hiding assumption of [29] has since found many applications. This suggests that the introduction and exploration of new assumptions, which we continue, is an interesting and productive line of research.

There is some feeling that “interactive” or “non-falsifiable” assumptions are undesirable. However, it depends on the particular assumption. There are interactive assumptions that are unbroken and successful, like OMDL [13], while many non-interactive ones have been broken. It is important that it be possible to show an assumption is false, but this is possible even for assumptions that are classified as “non-falsifiable;” for example, knowledge-of-exponent assumptions have successfully been shown to be false through cryptanalysis [16]. (The latter result assumes DL is hard.) MBDL is similarly amenable to cryptanalytic evaluation.

2.2 Preliminaries

**Notation.** If $n$ is a positive integer, then $\mathbb{Z}_n$ denotes the set $\{0, \ldots, n-1\}$ and $[n]$ or $[1..n]$ denote the set $\{1, \ldots, n\}$. We denote the number of coordinates of a vector $x$ by $|x|$. If $x$ is a vector then $|x|$ is its length (the number of its coordinates), $x[i]$ is its $i$-th coordinate and $[x] = \{x[i] : 1 \leq i \leq |x|\}$ is the set of all its coordinates. A string is identified with a vector over
\{0,1\}, so that if \(x\) is a string then \(x[i]\) is its \(i\)-th bit and \(|x|\) is its length. By \(\epsilon\) we denote the empty vector or string. The size of a set \(S\) is denoted \(|S|\). For sets \(D,R\) let \(\text{Func}(D,R)\) denote the set of all functions \(f:D \rightarrow R\).

Let \(S\) be a finite set. We let \(x \leftarrow S\) denote sampling an element uniformly at random from \(S\) and assigning it to \(x\). We let \(y \leftarrow A^{O_1,\ldots}(x_1,\ldots;r)\) denote executing algorithm \(A\) on inputs \(x_1,\ldots\) and coins \(r\) with access to oracles \(O_1,\ldots\) and letting \(y\) be the result. We let \(y \leftarrow A^{O_1,\ldots}(x_1,\ldots)\) be the resulting of picking \(r\) at random and letting \(y \leftarrow A^{O_1,\ldots}(x_1,\ldots;r)\). We let \([A^{O_1,\ldots}(x_1,\ldots)]\) denote the set of all possible outputs of \(A\) when invoked with inputs \(x_1,\ldots\) and oracles \(O_1,\ldots\). Algorithms are randomized unless otherwise indicated. Running time is worst case.

**GAMES.** We use the code-based game playing framework of [19]. (See Fig. 2.2 for an example.) Games have procedures, also called oracles. Amongst these are \texttt{Init} and a \texttt{Fin}. In executing an adversary \(\mathcal{A}\) with a game \(G_m\), procedure \texttt{Init} is executed first, and what it returns is the input to \(\mathcal{A}\). The latter may now call all game procedures except \texttt{Init, Fin}. When the adversary terminates, its output is viewed as the input to \texttt{Fin}, and what the latter returns is the game output. By \(\Pr[G_m(\mathcal{A})]\) we denote the event that the execution of game \(G_m\) with adversary \(\mathcal{A}\) results in output \(true\). In writing game or adversary pseudocode, it is assumed that boolean variables are initialized to false, integer variables are initialized to 0 and set-valued variables are initialized to the empty set \(\emptyset\). When adversary \(\mathcal{A}\) is executed with game \(G_m\), the running time of the adversary, denoted \(T_A\), assumes game procedures take unit time to respond. By \(Q_A^O\) we denote the number of queries made by \(\mathcal{A}\) to oracle \(O\) in the execution. These counts are all worst case.

**GROUPS.** Let \(G\) be a group of order \(p\). We will use multiplicative notation for the group operation, and we let \(1_G\) denote the identity element of \(G\). We let \(G^* = G \setminus \{1_G\}\) denote the set of non-identity elements, which is the set of generators of \(G\) if the latter has prime order. If \(g \in G^*\) is a generator and \(X \in G\), the discrete logarithm base \(g\) of \(X\) is denoted \(\text{DL}_{G,g}(X)\), and it is in the set \(\mathbb{Z}_{|G|}\).
2.3 The Multi-Base Discrete-Logarithm Problem

We introduce the multi-base discrete-logarithm (MBDL) problem. It is similar in flavor to the one-more discrete-logarithm (OMDL) problem [13], which has found many applications, in that it gives the adversary the ability to take discrete logarithms. For the rest of this Section, we fix a group \( G \) of prime order \( p = |G| \), and we fix a generator \( g \in G^* \) of \( G \). Recall that \( DL_{G,g} : G \rightarrow \mathbb{Z}_p \) is the discrete logarithm function in \( G \) with base \( g \).

**DL and OMDL.** We first recall the standard discrete logarithm (DL) problem via game \( G_{dl}^{dl} \) on the left of Figure 2.1. \( \text{INIT} \) provides the adversary, as input, a random challenge group element \( Y \), and to win it must output \( y' = DL_{G,g}(Y) \) to \( \text{FIN} \). We let \( \text{Adv}_{dl}^{dl}(\mathcal{A}) = \Pr[G_{dl}^{dl}(\mathcal{A})] \) be the discrete-log advantage of \( \mathcal{A} \).

In the OMDL problem [13], the adversary can obtain many random challenges \( Y_1, Y_2, \ldots, Y_n \in G \). It has access to a discrete log oracle that given \( W \in G \) returns \( DL_{G,g}(W) \). For better comparison with MBDL, let’s allow just one query to this oracle. To win it must compute the discrete logarithms of two group elements from the given list \( Y_1, Y_2, \ldots, Y_n \in G \). The integer \( n \geq 2 \) is a parameter of the problem.

---

**Figure 2.1:** Let \( G \) be a group of prime order \( p = |G| \), and let \( g \in G^* \) be a generator of \( G \). Left: Game defining standard discrete logarithm problem. Right: Game defining \((n,m)\)-multi-base discrete logarithm problem. Recall \( DL_{G,X}(W) \) is the discrete logarithm of \( W \in G \) to base \( X \in G^* \).
**MBDL.** In the MBDL problem we introduce, we return, as in DL, to there being a single random challenge point $Y$ whose discrete logarithm in base $g$ the adversary must compute. It has access to an oracle DLO to compute discrete logs, but rather than in base $g$ as in OMDL, to bases that are public, random group elements $X_1, X_2, \ldots, X_n$. It is allowed *just one* query to DLO. (As we will see, this is to avoid trivial attacks.) The integer $n \geq 1$ is a parameter of the problem.

Proceeding formally, consider game $G_{G, g, n}^\text{mbdl}$ on the right in Fig. 2.1, where $n \geq 1$ is an integer parameter called the number of bases. The adversary’s input, as provided by Init, is a random challenge group element $Y$ together with random generators $X_1, X_2, \ldots, X_n$. It can call oracle DLO with an index $i \in [n]$ and any group element $W \in G$ of its choice to get back $\text{DL}_{G, X_i}(W)$. Just one such call is allowed. At the end, the adversary wins the game if it outputs $y' = \text{DL}_{G, g}(Y)$ to Fin. We define the mbdl-advantage of $\mathcal{A}$ by

$$
\text{Adv}^\text{mbdl}_{G, g, n}(\mathcal{A}) = \Pr\left[G_{G, g, n}^\text{mbdl} (\mathcal{A})\right].
$$

**DISCUSSION.** By $n$-MBDL we will refer to the problem with parameter $n$. It is easy to see that if $n$-MBDL is hard then so is $n'$-MBDL for any $n' \leq n$. Thus, the smaller the value of $n$, the weaker the assumption. For our results, 1-MBDL, the weakest assumption in the series, suffices.

We explain why at most one DLO query is allowed. Suppose the adversary is allowed two queries. It could compute $a = \text{DLO}(1, Y) = \text{DL}_{G, X_1}(Y)$ and $b = \text{DLO}(1, g) = \text{DL}_{G, X_1}(g)$, so that $X_1^a = Y$ and $X_1^b = g$. Now the adversary returns $y' \leftarrow ab^{-1} \mod p$ and we have $g^{y'} = (g^{b^{-1}})^a = X_1^a = Y$, so the adversary wins.

As evidence for the hardness of MBDL, Theorem 2.5.1 proves good bounds on the adversary advantage in the generic group model (GGM). It is also important to consider non-generic approaches to the discrete logarithm problem over elliptic curves, including index-calculus methods and Semaev polynomials [82, 80, 83, 57, 49], but, to the best of our assessment, these
The MBDL problem as we have defined it can be generalized to allow multiple DLO queries with the restriction that at most one query is allowed per base, meaning for each \( i \) there can be at most one \( \text{DLO}(i, \cdot) \) query. In this paper, we do not need or use this extension. We have found applications based on it, but not pursued them because we have been unable to prove security of this extended version of MBDL in the GGM. We consider providing such a GGM proof an intriguing open question, resolving which would open the door to several new applications.

Our formalizations of DL and MBDL fix the generator \( g \). See [9] for a discussion of fixed versus random generators.

### 2.4 Schnorr Identification and Signatures from MBDL

In this section, we give a tight reduction of the IMP-PA security of the Schnorr identification scheme to the 1-MBDL problem and derive a corresponding improvement for Schnorr signatures.

**Identification Schemes.** We recall that a (canonical) identification scheme [1] ID
is a 3-move protocol in which the prover sends a first message called a commitment, the verifier sends a random challenge, the prover sends a response that depends on its secret key, and the verifier makes a decision to accept or reject based on the conversation transcript and the prover’s public key. Formally, ID is specified by algorithms ID.Kg, ID.Cmt, ID.Rsp, and ID.Vf, as well as a set ID.Chl of challenges Via \((pk, sk) \leftarrow s ID.Kg\), the key generation algorithm generates public verification key \(pk\) and associated secret key \(sk\). Algorithms ID.Cmt and ID.Rsp are the prover algorithms. The commitment algorithm ID.Cmt takes input the public key \(pk\) and returns a commitment message \(R\) to send to the verifier, as well as a state \(st\) for the prover to retain. The deterministic response algorithm ID.Rsp takes input the secret key \(sk\), a challenge \(c \in ID.Chl\) sent by the verifier, and a state \(st\), to return a response \(z\) to send to the verifier. The deterministic verification algorithm ID.Vf takes input the public key and a conversation transcript \(R, c, z\) to return a decision \(b \in \{true, false\}\) that is the outcome of the protocol.

An honest execution of the protocol is defined via procedure \(\text{Exec}_{ID}\) shown in the upper left of Fig. 2.2. It takes input a key pair \((pk, sk) \in [ID.Kg]\) to return a pair \((b, tr)\) where \(b \in \{true, false\}\) denotes the verifier’s decision whether to accept or reject and \(tr = (R, c, z)\) is the transcript of the interaction. We require that ID schemes satisfy \((perfect)\) completeness, namely that for any \((pk, sk) \in [ID.Kg]\) and any \((b, tr) \in [\text{Exec}_{ID}(sk, pk)]\) we have \(b = true\).

Impersonation under passive attack (IMP-PA) [44] is a security metric asking that an adversary not in possession of the prover’s secret key be unable to impersonate the prover, even given access to honestly generated transcripts. Formally, consider the game \(Gm_{imp-pa}^{ID}\) given in the right column of Fig. 2.2. An adversary has input the public key \(pk\) returned by INIT. It then has access to honest transcripts via the oracle Tr. When it is ready to convince the verifier, it submits its commitment \(R^*\) to oracle Ch. We allow only one query to Ch. In response the adversary obtains a random challenge \(c^*\). It must now output a response \(z^*\) to FIN, and the game returns true iff the transcript is accepted by ID.Vf. The \(R^*, c^*\) at line 4 are, respectively, the prior query
**Prover**

Input: \(X, x\)
- \(r \leftarrow \mathbb{Z}_p\)
- \(R \leftarrow g^r\)
- \(z \leftarrow (xc + r) \mod p\)

**Verifier**

Input: \(X\)
- \(c \leftarrow \mathbb{Z}_p\)
- \(b \leftarrow (g^z = RX^c)\)

<table>
<thead>
<tr>
<th>ID.Kg:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. (x \leftarrow \mathbb{Z}_p: X \leftarrow g^x); Return ((X, x))</td>
</tr>
<tr>
<td>2. (r \leftarrow \mathbb{Z}_p: R \leftarrow g^r); Return ((R, r))</td>
</tr>
<tr>
<td>3. (z \leftarrow (xc + r) \mod</td>
</tr>
<tr>
<td>4. Return (z)</td>
</tr>
<tr>
<td>5. (b \leftarrow (g^z = X^cR)); Return (b)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DS.Kg:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. (x \leftarrow \mathbb{Z}_p: X \leftarrow g^x)</td>
</tr>
<tr>
<td>2. Return ((X, x))</td>
</tr>
<tr>
<td>3. (r \leftarrow \mathbb{Z}_p: R \leftarrow g^r)</td>
</tr>
<tr>
<td>4. (c \leftarrow H(R, m))</td>
</tr>
<tr>
<td>5. (z \leftarrow (xc + r) \mod</td>
</tr>
<tr>
<td>6. Return ((R, z))</td>
</tr>
<tr>
<td>7. ((R, z) \leftarrow \sigma)</td>
</tr>
<tr>
<td>8. (c \leftarrow H(R, m))</td>
</tr>
<tr>
<td>9. Return ((g^z = X^cR))</td>
</tr>
</tbody>
</table>

**Figure 2.3:** Let \(G\) be a group of prime order \(p = |G|\) and let \(g \in G^*\) be a generator of \(G\). The Schnorr ID scheme \(ID = SchID[G, g]\) is shown pictorially at the top and algorithmically at the bottom left. At the bottom right is the Schnorr signature scheme \(DS = SchSig[G, g]\), using \(H: G \times \{0, 1\}^* \rightarrow \mathbb{Z}_p\).

to CH, and the response chosen at line 3. We define the IMP-PA advantage of \(\mathcal{A}\) against \(ID\) as

\[
\text{Adv}_{ID}^{\text{imp-pa}}(\mathcal{A}) = \Pr[\text{Gm}_{ID}^{\text{imp-pa}}(\mathcal{A})],
\]

the probability that the game returns true.

**Schnorr Identification Scheme and Prior Results.** Let \(G\) be a group of prime order \(p = |G|\), and \(g \in G^*\) a generator of \(G\). We recall the Schnorr identification scheme \([79]\) \(ID = SchID[G, g]\) in Fig. 2.3. The public key \(pk = X = g^x \in G\) where \(sk = x \in \mathbb{Z}_p\) is the secret key. The commitment is \(R = g^r \in G\), and \(r\) is returned as the prover state by the commitment algorithm. Challenges are drawn from \(ID.\text{Chl} = \mathbb{Z}_p\), and the response \(z\) and decision \(b\) are computed as shown.

The IMP-PA security of \(ID = SchID[G, g]\) based on DL is proven by a rewinding argument.
The simplest analysis is via the Reset Lemma of \cite{15}. It leads to the following (cf. \cite{15, Theorem 2}, \cite{17, Theorem 3}). Let $\mathcal{A}$ be an adversary attacking the IMP-PA security of $\text{ID}$. Then there is a discrete log adversary $\mathcal{B}$ such that

$$
\text{Adv}_{\text{ID}}^{\text{imp-pa}}(\mathcal{A}) \leq \sqrt{\text{Adv}_{G,g}^{\text{dl}}(\mathcal{B})} + \frac{1}{p}.
$$

Additionally, the running time $T_{\mathcal{B}}$ of $\mathcal{B}$ is roughly $2T_{\mathcal{A}}$ plus simulation overhead $O(Q_{\mathcal{T}_{A}} \cdot T_{G}^e)$, where $T_{G}^e$ is the time for an exponentiation in $G$.

**Our Result.** We show that the IMP-PA-security of the Schnorr identification scheme reduces tightly to the 1-MBDL problem. The reduction does not use rewinding. Our mbdl-adversary $\mathcal{B}$ solves the 1-MBDL problem by running the given imp-pa adversary $\mathcal{A}$ just once, so the mbdl-advantage, and running time, of the former, are about the same as the imp-pa advantage, and running time, of the latter. Refer to Section 3.2 for notation like $T_{\mathcal{A}}$, $Q_{\mathcal{T}_{A}}$.

**Theorem 2.4.1** Let $G$ be a group of prime order $p = |G|$, and let $g \in G^*$ be a generator of $G$. Let $\text{ID} = \text{SchID}[G, g]$ be the Schnorr identification scheme. Let $\mathcal{A}$ be an adversary attacking the imp-pa security of $\text{ID}$. Then we can construct an adversary $\mathcal{B}$ (shown explicitly in Figure 2.4) such that

$$
\text{Adv}_{\text{ID}}^{\text{imp-pa}}(\mathcal{A}) \leq \text{Adv}_{G,g,1}^{\text{mbdl}}(\mathcal{B}) + \frac{1}{p}.
$$

Additionally, $T_{\mathcal{B}}$ is roughly $T_{\mathcal{A}}$ plus simulation overhead $O(Q_{\mathcal{T}_{A}} \cdot T_{G}^e)$.

**Proof of Theorem 2.4.1:** Recall that, when reducing IMP-PA security of Schnorr to DL, the constructed dl adversary $\mathcal{B}$ sets the target point $Y$ to be the public key $X$. It is natural to take the same approach in our case. The question is how to use the discrete logarithm oracle DLO to avoid rewinding and get a tight reduction. But this is not clear and indeed the DLO oracle does not appear to help towards this.

Our reduction deviates from prior ones by not setting the target point $Y$ to be the public key $X$.
key. Instead we look at a successful impersonation by $\mathcal{A}$. (Simulation of $\mathcal{A}$’s transcript oracle Tr is again via the honest-verifier zero-knowledge property of the scheme.) Adversary $\mathcal{A}$ provides $R^*$, receives $c^*$ and then returns $z^*$ satisfying $g^{z^*} = R^*X^{c^*}$, where $X$ is the public key. Thus, $\mathcal{A}$ effectively computes the discrete logarithm of $R^*X^{c^*}$. We make this equal our mbdl challenge $Y$, meaning $B$, on input $Y$, arranges that $Y = R^*X^{c^*}$. If it can do this successfully, the $z^*$ returned by $\mathcal{A}$ will indeed be $DL_{G,g}(Y)$, which it can output and win.
But how can we arrange that $Y = R^*X^{c^*}$? This is where the DLO oracle enters. Adversary $B$ gives $X$ as input to $A$, meaning the public key is set to the group generator relative to which $B$ may compute discrete logarithms. Now, when $A$ provides $R^*$, our adversary $B$ returns a challenge $c^*$ that ensures $Y = R^*X^{c^*}$. This means $c^* = DL_{G,X}(YR^{*-1})$, and this is something $B$ can compute via its DLO oracle.

Some details include that the $X$ returned by \texttt{Init} is a generator, while the public key is a random group element, so they are not identically distributed, and that the challenge computed via DLO must be properly distributed. The analysis will address these.

For the formal proof, consider the games of Figure 2.4. Procedures indicate (via comments) in which games they are present. Game $G_{m1}$ includes the boxed code at line 2 while $G_{m0}$ does not. The games implement the transcript oracle via the zero-knowledge simulation rather than using the secret key, but otherwise $G_{m0}$ is the same as game $G_{imp-pa}^{ID}$ so we have

$$
Adv_{ID}^{imp-pa}(A) = Pr[G_{m0}(A)]
= Pr[G_{m1}(A)] + (Pr[G_{m0}(A)] - Pr[G_{m1}(A)]).
$$

Games $G_{m0}, G_{m1}$ are identical-until-bad, so by the Fundamental Lemma of Game Playing [19] we have

$$
Pr[G_{m0}(A)] - Pr[G_{m1}(A)] \leq Pr[G_{m1}(A) \text{ sets bad}] .
$$

Clearly $Pr[G_{m1}(A) \text{ sets bad}] \leq 1/p$. Now we can work with $G_{m1}$, where the public key $X$ is a random element of $\mathbb{G}^*$ rather than of $\mathbb{G}$. We claim that

$$
Pr[G_{m1}(A) = Pr[G_{m2}(A)].
$$

We now justify this. At line 4, game $G_{m2}$ picks $x$ directly from $\mathbb{Z}_p^*$, just like $G_{m1}$, and also
rewrites \textsc{Fin} in a different but equivalent way. The main thing to check is that \textsc{Ch} in \textsc{Gm2} is equivalent to that in \textsc{Gm1}, meaning line 6 results in \( c^* \) being uniformly distributed in \( \mathbb{Z}_p \). For this regard \( R^*, X \) as fixed and define the function \( f_{R^*, X} : \mathbb{G} \rightarrow \mathbb{Z}_p \) by \( f_{R^*, X}(Y) = \text{DL}_{G,X}(R^*-1Y) \). The adversary has no information about \( Y \) prior to receiving \( c^* \) at line 6, so the claim is established if we show that \( f_{R^*, X} \) is a bijection. This is true because \( X \in \mathbb{G}^* \) is a generator, which means that the function \( h_{R^*, X} : \mathbb{Z}_p \rightarrow \mathbb{G} \) defined by \( h_{R^*, X}(c^*) = R^*X^{c^*} \) is the inverse of \( f_{R^*, X} \). This establishes Eq. (2.9).

We now claim that adversary \( \mathcal{B} \), shown in Fig. 2.4, satisfies

\[
\Pr[\text{Gm2}(\mathcal{A})] \leq \text{Adv}^{\text{mbdl}}_{G, g, 1}(\mathcal{B}). \tag{2.10}
\]

Putting this together with the above completes the proof, so it remains to justify Eq. (2.10). Adversary \( \mathcal{B} \) has access to oracle DLO as per game \( G_{\text{mbdl}}^{G, g, 1} \). In the code, \textsc{Ch} and \textsc{Tr} are subroutines defined by \( \mathcal{B} \) and used to simulate the oracles of the same names for \( \mathcal{A} \). Adversary \( \mathcal{B} \) has input the challenge \( Y \) whose discrete logarithm in base \( g \) it needs to compute, as well as the base \( X \) relative to which it may perform one discrete log operation. It runs \( \mathcal{A} \) on input \( X \), so that the latter functions as the public key, which is consistent with \( \text{Gm2} \). The subroutine \textsc{Ch} uses DLO to produce \( c^* \) the same way as line 6 of \( \text{Gm2} \). It simulates \textsc{Tr} as per line 7 of \( \text{Gm2} \). If \( \text{Gm2} \) returns true at line 9 then we have \( g^{z^*} = X^{c^*}R^* = WR^* = R^*Y^{R^*} = Y \), so \( \mathcal{B} \) wins.}

**Quantitative Comparison.** Concrete security improvements are in the end efficiency improvements, because, for a given security level, we can use smaller parameters, and thus the scheme algorithms are faster. Here we quantify this, seeing what Eq. (2.8) buys us over Eq. (2.7) in terms of improved efficiency for the identification scheme.

We take as goal to ensure that any adversary \( \mathcal{A} \) with running time \( t \) has advantage

\[
\text{Adv}^{\text{imp-pa}}_{\text{ID}}(\mathcal{A}) \leq \epsilon
\]

in violating IMP-PA security of \( \text{ID} = \text{SchID}[G, g] \). Here \( t, \epsilon \) are parameters for which many choices are possible. For example, \( t = 2^{90} \) and \( \epsilon = 2^{-32} \) is one choice, reflecting a
128-bit security level, where we define the bit-security level as $\log_2(t/\epsilon)$. The cost of scheme algorithms is the cost of exponentiation in the group, which is cubic in the representation size $k = \log p$ of group elements. So we ask what $k$ must be to provably ensure the desired security. Equations (2.7) and (2.8) will yield different choices of $k$, denoted $k_1$ and $k_2$, with $k_2 < k_1$. We will conclude that Eq. (2.8) allows a $s = (k_1/k_2)^3$-fold speedup for the scheme.

Let $\mathcal{B}_1$ denote the DL adversary referred to in Eq. (2.7), and $\mathcal{B}_2$ the 1-MBDL adversary referred to in (2.8). To use the equations, we now need estimates on their respective advantages. For this, we assume $G$ is a group in which the security of discrete-log-related problems is captured by the bounds proven in the generic group model (GGM), as seems to be true, to best of our current understanding, for certain elliptic curve groups. We will ignore the simulation overhead in running time since the number of transcript queries of $\mathcal{A}$ reflects online executions of the identification protocol and should be considerably less than the running time of $\mathcal{A}$, so that we take the running times of both $\mathcal{B}_1$ and $\mathcal{B}_2$ to be about $t$, the running time of our IMP-PA adversary $\mathcal{A}$. Now the classical result of Shoup [81] says that $\text{Adv}_{\text{dl}}^{\text{G}}(\mathcal{B}_1) \approx t^2/p$, and our Theorem 2.5.1 says that also $\text{Adv}_{\text{mbdl}}^{\text{G}}(\mathcal{B}_2) \approx t^2/p$.

Here we pause to highlight that these two bounds being the same is a central attribute of the 1-MBDL assumption. That Theorem 2.4.1 (as per Figure 2.1) provides efficiency improvements stems not just from the reduction of Eq. (2.8) being tight, but also from that fact that the 1-MBDL problem is just as hard to solve as the DL problem, meaning $\text{Adv}_{\text{mbdl}}^{\text{G}}(\mathcal{B}_2) \approx \text{Adv}_{\text{dl}}^{\text{G}}(\mathcal{B}_1) \approx t^2/p$.

Continuing, putting together what we have so far gives two bounds on the IMP-PA advantage of $\mathcal{A}$, the first via Equations (2.7) and the second via Eq. (2.8), namely, dropping the $1/p$ terms,

$$\text{Adv}_{\text{id}}^{\text{imp-pa}}(\mathcal{A}) \leq \epsilon_1(t) = \sqrt{\frac{t^2}{p}} = \frac{t}{\sqrt{p}} \tag{2.11}$$

$$\text{Adv}_{\text{id}}^{\text{imp-pa}}(\mathcal{A}) \leq \epsilon_2(t) = \frac{t^2}{p} \tag{2.12}$$
Recall our goal was to ensure that $\text{Adv}_{\text{SchID}[G,R]}^{\text{imp-pa}}(A) \leq \epsilon$. We ask, what value of $p$, in either case, ensures this? Solving for $p$ in the equations $\epsilon = \epsilon_1(t)$ and $\epsilon = \epsilon_2(t)$, we get two corresponding values, namely $p_1 \approx t^2/\epsilon^2$ and $p_2 \approx t^2/\epsilon$. We see that $p_1 > p_2$, meaning Theorem 2.4.1 guarantees the same security as Eq. (2.7) in groups of a smaller size. Finally, the ratio of representation sizes for group elements is

$$r \approx \frac{\log(p_1)}{\log(p_2)} \approx \frac{\log(t^2/\epsilon) + \log(1/\epsilon)}{\log(t^2/\epsilon)} = 1 + \frac{\log(1/\epsilon)}{\log(t^2/\epsilon)}.$$

Scheme algorithms employ exponentiation in the group and are thus cubic time, so the ratio of speeds is $s = r^3$, which we call the speedup factor, and we can now estimate it numerically. For a few values of $t, \epsilon$, Figure 2.1 shows the log of the group size $p_i$ needed to ensure the desired security under prior results ($i = 1$) and ours ($i = 2$). Then it shows the speedup $s$. For example if we want attacks of time $t = 2^{64}$ to achieve advantage at most $\epsilon = 2^{-64}$, prior results would require a group of size $p_1$ satisfying $\log(p_1) \approx 256$, while our results allow it with a group of size $\log(p_2) \approx 192$, which yields a 2.4x speedup. Of course many more examples are possible.

**SIGNATURE SCHEMES.** Towards results on the Schnorr signature scheme, we start by recalling definitions. A signature scheme $\text{DS}$ specifies key generation algorithm $\text{DS.Kg}$, signing algorithm $\text{DS.Sign}$, deterministic verification algorithm $\text{DS.Vf}$ and a set $\text{DS.HF}$ of functions called the hash function space. Via $(pk, sk) \leftarrow \text{DS.Kg}$ the signer generates a public verification key $pk$ and secret signing key $sk$. Via $\sigma \leftarrow \text{DS.Sign}^h(sk, m)$ the signing algorithm takes $sk$ and a message $m \in \{0, 1\}^*$, and, with access to an oracle $h \in \text{DS.HF}$, returns a signature $\sigma$. Via $b \leftarrow \text{DS.Vf}^h(pk, m, \sigma)$, the verifier obtains a boolean decision $b \in \{\text{true}, \text{false}\}$ about the validity of the signature. The correctness requirement is that for all $h \in \text{DS.HF}$, all $(pk, sk) \in [\text{DS.Kg}]$, all $m \in \{0, 1\}^*$ and all $\sigma \in [\text{DS.Sign}^h(sk, m)]$ we have $\text{DS.Vf}^h(pk, m, \sigma) = \text{true}$.

Game $G_{\text{uf}}$ in Fig. 2.5 captures UF (unforgeability under chosen-message attack) [50]. Procedure $H$ is the random oracle [18], implemented as a function $h$ chosen at random from
Game $G_{DS}^{uf}$

**INIT:**
1. $h \leftarrow DS.HF$ ; $(pk, sk) \leftarrow DS.Kg$
2. Return $pk$

**SIGN($m$):**
3. $\sigma \leftarrow DS.Sign^H(sk, m)$ ; $S \leftarrow S \cup \{m\}$
4. Return $\sigma$

**H($x$):**
5. Return $h(x)$

**FIN($m_*, \sigma_*$):**
6. Return (($m_* \not\in S$) and $DS.Vf^H(pk, m_*, \sigma_*)$)

**Figure 2.5:** Game defining UF security of signature scheme $DS$.

$DS.HF$. We define the UF advantage of adversary $A$ as $\text{Adv}_{DS}^{uf}(A) = \Pr[G_{DS}^{uf}(A)]$.

**Schnorr Signatures.** The Schnorr signature scheme $DS = \text{SchSig}[[G, g]$ is derived by applying the Fiat-Shamir transform [45] to the Schnorr identification scheme. Its algorithms are shown at the bottom right of Fig. 2.3. The set $DS.HF$ consists of all functions $h : G \times \{0, 1\}^* \rightarrow \mathbb{Z}_p$.

**Our and Prior Results.** We give a reduction, of the UF security of the Schnorr signature scheme to the 1-MBDL problem, that loses only a factor of the number of hash-oracle queries of the adversary. We start by recalling the following lemma from [1]. It derives the UF security of $\text{SchSig}[G, g]$ from the IMP-PA security of $\text{SchID}[G, g]$:

**Lemma 2.4.2** [1] Let $G$ be a group of prime order $p = |G|$, and let $g \in G^*$ be a generator of $G$. Let $\text{ID} = \text{SchID}[G, g]$ and $DS = \text{SchID}[G, g]$ be the Schnorr identification and signature schemes, respectively. Let $A_{ds}$ be an adversary attacking the uf-security of $DS$. Let $\alpha = (1 + Q^H_{A_{ds}} + Q^S_{A_{ds}})Q^S_{A_{ds}}$. Then we can construct an adversary $A_{id}$ such that

$$\text{Adv}_{DS}^{uf}(A_{ds}) \leq (1 + Q^H_{A_{ds}}) \cdot \text{Adv}_{ID}^{\text{imp-pa}}(A_{id}) + \frac{\alpha}{p}.$$ 

Additionally, $T_{A_{id}} \approx T_{A_{ds}}$ and $Q^T_{A_{id}} = Q^S_{A_{ds}}$. 

86
Combining this with Theorem 2.4.1, we have:

**Theorem 2.4.3** Let $G$ be a group of prime order $p = |G|$, and let $g \in G^*$ be a generator of $G$. Let $DS = \text{SchSig}[G, g]$ be the Schnorr signature scheme. Let $A$ be an adversary attacking the uf security of $ID$. Let $\beta = (1 + Q^H_A + Q^{\text{SIGN}}_A)Q^{\text{SIGN}}_A + (1 + Q^H_A)$. Then we can construct an adversary $B$ such that

$$\text{Adv}_{DS}(A) \leq (1 + Q^H_A) \cdot \text{Adv}_{G,G,1}(B) + \frac{\beta}{p}.$$  \hfill (2.13)

Additionally, $T_B$ is roughly $T_A$ plus simulation overhead $O(Q^{\text{SIGN}}_A \cdot T_G^e)$.

Let’s compare this to prior results. A simple proof of UF-security of $DS$ from DL can be obtained by combining Lemma 2.4.2 with the classical DL-based security of $ID$ as given by Eq. (2.7). For $A$ an adversary attacking the UF security of $DS$, this would yield a discrete log adversary $B$ such that

$$\text{Adv}_{DS}(A) \leq (1 + Q^H_A) \cdot \sqrt{\text{Adv}_{G,G,1}(B)} + \frac{\beta}{p},$$  \hfill (2.14)

where $\beta$ is as in Theorem 2.4.3 and $T_B$ is about $2T_A$ plus the same simulation overhead as above. This is however *not* the best prior bound. One can do better with a direct application of the general Forking Lemma of [14] as per [76]. For $A$ an adversary attacking the UF security of $DS$, this would yield a discrete log adversary $B$ such that

$$\text{Adv}_{DS}(A) \leq \sqrt{(1 + Q^H_A) \cdot \text{Adv}_{G,G,1}(B)} + \frac{\beta}{p},$$  \hfill (2.15)

where $\beta$ and $T_B$ are as above. The reason Eq. (2.15) is a better bound than Eq. (2.14) is that the $1 + Q^H_A$ term has moved under the square root. Still we see that Eq. (2.13) is even better; roughly (neglecting the additive term), the bound in Eq. (2.13) is the square of the one in Eq. (2.15), and thus (always) smaller.

**Quantitative Comparisons.** Our numerical comparisons will be with the best prior bound, meaning that of Eq. (2.15). For a few values of $t, q_h, \varepsilon$ with $t \geq q_h = Q^H_A$, Figure 2.1 shows
the speedup $s$ from Eq. (2.13) over Eq. (2.15). The table shows that the speedup is a bit less than for Schnorr identification shown in the same Figure, but still significant. For example if we want attacks of time $t = 2^{64}$ to achieve advantage at most $\epsilon = 2^{-64}$, Theorem 2.4.3 is allowing group sizes to go down enough to yield a 5.4-fold speedup.

To derive these estimates, we use the same framework and setup as we did for identification. Let $\mathbb{G}$ be a group of prime order $p$ with generator $g$. We take as goal to ensure that any adversary $A$ with running time $t$, making $q_h$ queries to $H$ and $q_s$ queries to $\text{SIGN}$, has advantage $\text{Adv}_{\text{DS}}^\text{uf}(A) \leq \epsilon$ in violating UF security of $\text{DS} = \text{SchSig}[\mathbb{G},g]$, where $t, \epsilon, q_h, q_s$ are parameters. We assume $q_s < q_h \leq t$, as one expects in practice. Let $B_1, B_2$ be the adversaries of Equations (2.15) and (2.13), respectively. As before, assume $\text{Adv}_{\mathbb{G},g}^\text{dl}(B_1) \approx t^2/p$ from [81], and also $\text{Adv}_{\mathbb{G},g}^\text{mbl}(B_2) \approx t^2/p$ from Theorem 2.5.1. Then

\[
\text{Adv}_{\text{DS}}^\text{uf}(A) \leq \epsilon_1(t, q_h) \approx \sqrt{\frac{q_h t^2}{p}} \\
\text{Adv}_{\text{DS}}^\text{uf}(A) \leq \epsilon_2(t, q_h) \approx q_h \frac{t^2}{p} = \frac{q_h t^2}{p} \approx \epsilon_1(t, q_h)^2.
\]

In the estimates above, we have dropped the additive term, which has order $q_h q_s / p$, because this is negligible compared to the other term for reasonable parameter values, including the ones we consider. This leaves $\epsilon_1, \epsilon_2$ not depending on $q_s$, but recall the latter is expected to be (much) smaller than $q_h$. Then our bound $\epsilon_2$ is about the square of the prior one, and thus always smaller.

We now ask what value of $p$ ensures $\text{Adv}_{\text{DS}}^\text{uf}(A) \leq \epsilon$, in each case. Solving $\epsilon_1(t, q_h) \leq \epsilon$ yields $p_1 \approx t^2 q_h / \epsilon^2$, and solving $\epsilon_2(t, q_h) \leq \epsilon$ yields $p_2 \approx t^2 q_h / \epsilon$. As before we see that $p_2 < p_1$, meaning Theorem 2.4.1 guarantees security in groups of smaller size. The ratio of the representation-size of group elements is

\[
r \approx \frac{\log(p_1)}{\log(p_2)} \approx \frac{\log(t^2 q_h / \epsilon) + \log(1/\epsilon)}{\log(t^2 q_h / \epsilon)} = 1 + \frac{\log(1/\epsilon)}{\log(t^2 q_h / \epsilon)}.
\]
Game $G_{G,n}^{\text{gg-mbdl}}$ I Set $G$ is the range of the encoding, $|G|$ is prime.

INIT():
1. $p \leftarrow |G|$ ; $E \leftarrow \text{Bijections}(Z_p, G)$ I Think “$E(x) = g^x$”
2. $1 \leftarrow E(0)$ ; $g \leftarrow E(1)$ I Think $1$ the identity and $g$ generator
3. $y \leftarrow Z_p$ ; $Y \leftarrow E(y)$
4. For $i = 1, \ldots, n$ do $x_i \leftarrow Z_p$ ; $X_i \leftarrow E(x_i)$
5. $GL \leftarrow \{1, g, Y, X_1, \ldots, X_n\}$
6. Return $1, g, Y, X_1, \ldots, X_n$

OP$(A, B, \text{sgn})$: $A, B \in G$ and $\text{sgn} \in \{+, -\}$
7. If $(A \notin GL$ or $B \notin GL$) then return $\perp$
8. $c \leftarrow (E^{-1}(A) \text{sgn} E^{-1}(B)) \mod p$ ; $C \leftarrow E(c)$ ; $GL \leftarrow GL \cup \{C\}$
9. Return $C$

DLO$(i, W)$: $i \in [1..n]$ and $W \in G$
10. If $(W \notin GL)$ then return $\perp$
11. $z \leftarrow x_i^{-1} \cdot E^{-1}(W) \mod p$ I $x_i^{-1}$ is inverse of $x_i$ mod $p$
12. Return $z$

FIn$(y')$:
13. Return $(y = y')$

Figure 2.6: Game defining $n$-MBDL problem in the generic group model.

As before the ratio of speeds (speedup factor) is $s = r^3$, and we can now estimate it numerically. For a few values of $t, \varepsilon$, Figure 2.1 shows the log of the group size $p_i$ needed to ensure the desired security under prior results ($i = 1$) and ours ($i = 2$). Then it shows the speedup $s$.

2.5 MBDL hardness in the Generic Group Model

With a new problem like MBDL it is important to give evidence of hardness. Here we provide this in the most common and accepted form, namely a proof of hardness in the generic group model (GGM).

The quantitative aspect of the result is just as important as the qualitative. Theorem 2.5.1 below says that the advantage of a GGM adversary $A$ in breaking $n$-MBDL is $O(q^2/p)$ where $q$ is $n$ plus the number of group operations (time) invested by $A$, namely about the same as the
ggm-dl-advantage of an adversary of the same resources. Reductions (to some problem) from MBDL that are tighter than ones from DL now bear fruit in justifying the secure use of smaller groups, which lowers costs.

The proof of Theorem 2.5.1 begins with a Lemma that characterizes the distribution of replies to the DLO query. A game sequence is then used to reduce bounding the adversary advantage to some static problems in linear algebra.

Some prior proofs in the GGM have been found to be wrong. (An example is that of [22] as pointed out by [54]. We note that the assumption was changed to fill the gap in [23].) Also we, at least, have often found GGM proofs imprecise and hard to verify. This has motivated us to try to be precise with definitions and to attend to details.

Starting with definitions, we associate to any encoding function $E$ an explicit binary operation $\circ_E$ that turns the range-set of $E$ into a group. A random choice of $E$ then results in the GGM, with the “generic group” being now explicitly defined as the group associated to $E$. The proof uses a game sequence and has been done at a level of detail that is perhaps unusual in this domain.

**MBDL IN THE GGM.** We start with definitions. Suppose $G$ is a set whose size $p = |G|$ is a prime, and $E : \mathbb{Z}_p \to G$ is a bijection, called the encoding function. For $A, B \in G$, define $A \circ_E B = E(E^{-1}(A) + E^{-1}(B))$. Then $G$ is a group under the operation $\circ_E$ [86], with identity element $E(0)$, and the encoding function becomes a group homomorphism: $E(a + b) = E(a) \circ_E E(b)$ for all $a, b \in \mathbb{Z}_p$. The element $g = E(1) \in G$ is a generator of this group, and $E^{-1}(A)$ is then the discrete logarithm of $A \in G$ relative to $g$. We call $\circ_E$ the group operation on $G$ induced by $E$.

In the GGM, the encoding function $E$ is picked at random and the adversary is given an oracle for the group operation $\circ_E$ induced on $G$ by $E$. Game $G_{G,n}^{gg\text{-}mbdl}$ in Fig. 2.6 defines, in this way, the $n$-MBDL problem. The set $G$ parameterizes the game, and the random choice of encoding function $E : \mathbb{Z}_p \to G$ is shown at line 1. Procedure $\text{Op}$ then implements either the group operation $\circ_E$ on $G$ induced by $E$ (when $\text{sgn}$ is $+$) or its inverse (when $\text{sgn}$ is $-$). Lines 3,4
pick \( y, x_1, \ldots, x_n \) and define the corresponding group elements \( Y, X_1, \ldots, X_n \). Set GL holds all group elements generated so far. The new element here is the oracle DLO that takes \( i \in [1..n] \) and \( W \in G \) to return the discrete logarithm of \( W \) in base \( X_i \). This being \( x_i^{-1} \) times the discrete logarithm of \( W \) in base \( g \), the procedure returns \( z \leftarrow x_i^{-1} \cdot E^{-1}(W) \). The inverse and the operations here are modulo \( p \). Only one query to this oracle is allowed, and the adversary wins if it halts with output \( y' \) that equals \( y \). We let \( \text{Adv}_{G,n}^{gg-\text{mbdl}}(A) = \Pr[G_{G,n}^{gg-\text{mbdl}}(A)] \) be its ggm-\text{mbdl}-advantage.

**RESULT.** The following upper bounds the ggm-\text{mbdl}-advantage of an adversary \( A \) as a function of the number of its \( \text{Op} \) queries and \( n \).

**Theorem 2.5.1** Let \( G \) be a set whose size \( p = |G| \) is a prime. Let \( n \geq 1 \) be an integer. Let \( A \) be an adversary making \( Q_{A}^{\text{Op}} \) queries to its \( \text{Op} \) oracle and one query to its DLO oracle. Let \( q = Q_{A}^{\text{Op}} + n + 3 \). Then

\[
\text{Adv}_{G,n}^{gg-\text{mbdl}}(A) \leq \frac{2 + q(q - 1)}{p - 1}.
\] (2.16)

**PROOF FRAMEWORK AND LEMMA.** Much of our work in the proof is over \( \mathbb{Z}_{p}^{n+2} \) regarded as a vector space over \( \mathbb{Z}_p \). We let \( \vec{0} \in \mathbb{Z}_{p}^{n+2} \) be the all-zero vector, and \( \vec{e}_i \in \mathbb{Z}_{p}^{n+2} \) the \( i \)-th basis vector, meaning it has a 1 in position \( i \) and zeros elsewhere. We let \( \langle \vec{a}, \vec{b} \rangle = (\vec{a}[1]\vec{b}[1] + \cdots + \vec{a}[n + 2]\vec{b}[n + 2]) \) denote the inner product of vectors \( \vec{a}, \vec{b} \in \mathbb{Z}_{p}^{n+2} \), where the operations are modulo \( p \).

In the GGM, the encoding function takes as input a point in \( \mathbb{Z}_p \). The proof of GGM hardness of the DL problem [81] moved to a modified encoding function that took input a univariate polynomial, the variable representing the target discrete logarithm \( y \). We extend this to have the modified encoding function take input a degree one polynomial in \( n + 1 \) variables, these representing \( x_1, \ldots, x_n, y \). The polynomial will be represented by the vector of its coefficients, so that representations, formally, are vectors in \( \mathbb{Z}_{p}^{n+2} \). At some point, games in our proof will need to simulate the reply to a DLO\((i, W) \) query, meaning provide a reply \( z \) without knowing \( x_i \). At this point, \( W \in G \) will be represented by a vector \( \vec{w} \in \mathbb{Z}_{p}^{n+2} \) that is known to the game and adversary.
The natural simulation approach is to return a random $z \leftarrow \mathbb{Z}_p$ or $z \leftarrow \mathbb{Z}_p^*$, but these turn out to not perfectly mimic the true distribution of replies, because this distribution depends on $\vec{w}$. We start with a lemma that describes how to do a perfect simulation.

While the above serves as motivation for the Lemma, the Lemma itself is self-contained, making no reference to the DLO oracle. We consider the games of Figure 2.7. They are played with an adversary making a single Init query whose arguments are an integer $i \in [1..n]$ and a vector $\vec{w} \in \mathbb{Z}_p^{n+2}$. The operations in the games, including inverses of elements in $\mathbb{Z}_p^*$, are in the field $\mathbb{Z}_p$. Game $G_{p,n}^{\text{simp}-1}$ captures what, in our vector-representation, will be the “real” game, with $z$ at line 3 computed correctly as a function of $x_i$. Game $G_{p,n}^{\text{simp}-0}$ represents the simulation, first picking $z$ and then defining $x_i$. Lines 3,4 show that there are two cases for how $z,x_i$ are chosen in the simulation, depending on the value of $w = \vec{w}[i]$ and the inner product of $\vec{w}$ with $\vec{x}_i$. The games return all variables involved. The claim is that the outputs of the games are identically distributed, captured formally, in the statement of Lemma 2.5.2 below, as the condition that any adversary returns true with the same probability in the two games.

**Figure 2.7: Games for Lemma 2.5.2**

The natural simulation approach is to return a random $z \leftarrow \mathbb{Z}_p$ or $z \leftarrow \mathbb{Z}_p^*$, but these turn out to not perfectly mimic the true distribution of replies, because this distribution depends on $\vec{w}$. We start with a lemma that describes how to do a perfect simulation.

While the above serves as motivation for the Lemma, the Lemma itself is self-contained, making no reference to the DLO oracle. We consider the games of Figure 2.7. They are played with an adversary making a single Init query whose arguments are an integer $i \in [1..n]$ and a vector $\vec{w} \in \mathbb{Z}_p^{n+2}$. The operations in the games, including inverses of elements in $\mathbb{Z}_p^*$, are in the field $\mathbb{Z}_p$. Game $G_{p,n}^{\text{simp}-1}$ captures what, in our vector-representation, will be the “real” game, with $z$ at line 3 computed correctly as a function of $x_i$. Game $G_{p,n}^{\text{simp}-0}$ represents the simulation, first picking $z$ and then defining $x_i$. Lines 3,4 show that there are two cases for how $z,x_i$ are chosen in the simulation, depending on the value of $w = \vec{w}[i]$ and the inner product of $\vec{w}$ with $\vec{x}_i$. The games return all variables involved. The claim is that the outputs of the games are identically distributed, captured formally, in the statement of Lemma 2.5.2 below, as the condition that any adversary returns true with the same probability in the two games.
Lemma 2.5.2 Let $p$ be a prime and $n \geq 1$ an integer. Then for any adversary $\mathcal{A}$ we have

$$\Pr[G_{p,n}^{\text{smp}-1}(\mathcal{A})] = \Pr[G_{p,n}^{\text{smp}-0}(\mathcal{A})],$$

(2.17)

where the games are in Figure 2.7

Proof of Lemma 2.5.2: With $\bar{w}, \bar{x}$ being $\mathcal{A}$’s query to $\text{Init}$, we can regard vector $\bar{x}_i = (x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n, 1, y)$ as fixed, since its constituents are chosen identically in the two games. Let $\alpha = (\bar{w}, \bar{x}_i)$. Now consider two cases. The first is that $\alpha = 0$. Then, in both games, $z = w$, and $x_i$ is chosen randomly from $\mathbb{Z}_p^*$. The second case is that $\alpha \neq 0$. For $x \in \mathbb{Z}_p^*$ let $Z_{w,\alpha}(x) = w + x^{-1} \cdot \alpha$, so that $z = Z_{w,\alpha}(x_i)$ at line 3 of game $G_{p,n}^{\text{smp}-1}$. That $\alpha \neq 0$ implies $Z_{w,\alpha}(x) \neq w$, meaning the function $Z_{w,\alpha}$ maps as $Z_{w,\alpha} : \mathbb{Z}_p^* \to \mathbb{Z}_p \setminus \{w\}$. For $z \in \mathbb{Z}_p \setminus \{w\}$, let $X_{w,\alpha}(z) = \alpha \cdot (z - w)^{-1}$, so that $x_i = X_{w,\alpha}(z)$ at line 3 of game $G_{p,n}^{\text{smp}-0}$. That $z \neq w$ and $\alpha \neq 0$ means $X_{w,\alpha}(z) \neq 0$, meaning the function $X_{w,\alpha}$ maps as $X_{w,\alpha} : \mathbb{Z}_p \setminus \{w\} \to \mathbb{Z}_p^*$. The proof is complete if we show that these functions are inverses of each other, in particular showing that both are bijections. Indeed, for any $x \in \mathbb{Z}_p^*$ we have $X_{w,\alpha}(Z_{w,\alpha}(x)) = X_{w,\alpha}(w + x^{-1} \cdot \alpha) = \alpha \cdot (w + x^{-1} \cdot \alpha - w)^{-1} = \alpha \cdot x \cdot \alpha^{-1} = x$. \]

Equipped with this lemma, we give the proof of Theorem 2.5.1.

Proof of Theorem 2.5.1: By $\text{span}(\bar{v})$ we denote the span of a vector $\bar{v} \in \mathbb{Z}_p^{n+2}$, which simply means the set of all $a \cdot \bar{v}$ as $a$ ranges over $\mathbb{Z}_p$. Beyond the procedures of game $G_{G,n,m}^{\text{gg-mbdl}}$, some of our games define procedures $\text{VE}$ and $\text{VE}^{-1}$, the vector-encoding and its inverse. These procedures are not exported, meaning can be called only by other game procedures, not by the adversary. Throughout, we assume the adversary $\mathcal{A}$ makes no trivial queries. By this we mean that the checks at lines 7 and 10 of game $G_{G,n,m}^{\text{gg-mbdl}}$ are not triggered. In our games the consequence is that we assume $\text{TI}[A], \text{TI}[B] \neq \bot$ in any $\text{Op}(A, B, \text{sgn})$ query and, for a $\text{DLO}(i, W)$ query, that $i \in [n]$, that $\text{TI}[W] \neq \bot$ and that the number of queries to this oracle is exactly $m = 1$. (The table $\text{TI}[\cdot]$ referred to here starts appearing in Game $G_m$ of Figure 2.8.)
**Figure 2.8**: Game $G_{m_0}$ for the proof of Theorem 2.5.1. Some procedures will also be in later games, as marked.

We start with game $G_{m_0}$ of Figure 2.8, claiming that

$$\text{Adv}_{G,n,m}^{gg-mdl}(A) = \Pr[G_{m_0}(A)].$$

(2.18)

We now explain the game and justify Eq. (2.18). At line 10, operation $\text{sgn}$ is performed modulo $p$, and at line 11, the inverse and product in computing $z$ are modulo $p$. The game picks $y, x_1, \ldots, x_n$ in the same way as game $G_{G,n,m}^{gg-mdl}$. At line 1, it also picks encoding function $E$ in the same way as game $G_{G,n,m}^{gg-mdl}$, but does not use this function directly to do the encoding, instead calling $\text{VE}$, which we call the vector-encoding function, on the indicated vector arguments. This procedure maintains tables $TV : \mathbb{Z}_p^{n+2} \to G \cup \{\perp\}$ and $\text{TI} : G \to \mathbb{Z}_p^{n+2} \cup \{\perp\}$ (the “I” stands for “inverse”) that from the code can be seen to satisfy the following, where vector $\vec{x}$ is defined at line 3:
VE(\vec{r}) \colon I_{Gm_1} G_{m_2}. Here \vec{r} \in \mathbb{Z}_p^{n+2}.

13 \quad \text{If (TV[\vec{r}] \neq \bot) then return TV[\vec{r}]

14 \quad \text{If (\exists \vec{r}^\prime : (TV[\vec{r}^\prime] \neq \bot \text{ and } \vec{r} - \vec{r}^\prime \in \text{span}(\vec{v})) then}

15 \quad C \leftarrow TV[\vec{r}^\prime]; \text{TV}[\vec{r}] \leftarrow C; \text{TI}[C] \leftarrow \vec{r}; \text{Return TV[\vec{r}]}

16 \quad C \leftarrow G \setminus \text{GL}

17 \quad \text{If (\exists \vec{r}^\prime : (TV[\vec{r}] \neq \bot \text{ and } \langle \vec{r}, \vec{x} \rangle = \langle \vec{r}^\prime, \vec{x} \rangle ) then}

18 \quad \text{bad} \leftarrow \text{true}; C \leftarrow TV[\vec{r}^\prime]

19 \quad \text{TV}[\vec{r}] \leftarrow C; \text{TI}[C] \leftarrow \vec{r}; \text{GL} \leftarrow \text{GL} \cup \{C\}; \text{Return TV[\vec{r}]}

DLO(i, W) \colon I_{Gm_1}, G_{m_2}. Here i \in [n] and TI[W] \neq \bot.

20 \quad \vec{w} \leftarrow \text{VE}^{-1}(W); z \leftarrow (x_i)^{-1} \cdot \langle \vec{w}, \vec{x} \rangle; \vec{v} \leftarrow \vec{w} - z \cdot \vec{e}_i

21 \quad \text{Return } z

**Figure 2.9:** Procedures for games $G_{m_1}, G_{m_2}, G_{m_3}$ in the proof of Theorem 2.5.1 where $G_{m_1}$ includes the boxed code.

1. If $TV[\vec{r}] \neq \bot$ then $TV[\vec{r}] = E(\langle \vec{r}, \vec{x} \rangle)$
2. If $\text{TI}[C] \neq \bot$ then $\langle \text{TI}[C], \vec{x} \rangle = E^{-1}(C)$

This ensures Eq. (2.18) as follows. From line 4 and the above we have $g = TV[\vec{e}_{n+1}] = E(\langle \vec{e}_{n+1}, \vec{x} \rangle) = E(1)$, and, similarly, we have $Y = E(y)$ and $X_i = E(x_i)$ for $i \in [1..n]$, meaning these quantities are as in game $G^{gg,mbdl}_{G,n,m}$. Turning to OP, by linearity of the inner product and item (2) above, we have

\[
\langle \vec{c}, \vec{x} \rangle = \langle \text{TI}[A] \text{sgn} \text{TI}[B], \vec{x} \rangle = \langle \text{TI}[A], \vec{x} \rangle \text{ sgn} \langle \text{TI}[B], \vec{x} \rangle
\]

\[= E^{-1}(A) \text{ sgn} E^{-1}(B),\]

so by item (1) we have $\text{VE}(\vec{c}) = E(E^{-1}(A) \text{ sgn} E^{-1}(B))$, as in game $G^{gg,mbdl}_{G,n,m}$. Finally, for DLO, item (2) says that $\langle \vec{w}, \vec{x} \rangle = E^{-1}(W)$, again as in game $G^{gg,mbdl}_{G,n,m}$.

Games $G_{m_1}, G_{m_2}$ are formed by taking the indicated procedures of Figure 2.8 and adding those of Figure 2.9 with the former game including the boxed code, and the latter not. Procedure VE no longer invokes $E$, instead sampling it lazily. The vector $\vec{v}$ defined at line 20 satisfies $\langle \vec{v}, \vec{x} \rangle = \langle \vec{w} - z \cdot \vec{e}_i, \vec{x} \rangle = \langle \vec{w}, \vec{x} \rangle - z \cdot \langle \vec{e}_i, \vec{x} \rangle = \langle \vec{w}, \vec{x} \rangle - x_i^{-1} \cdot \langle \vec{w}, \vec{x} \rangle \cdot x_i = 0$. As a result, at any time,
any vector $\vec{u} \in \text{span}(\vec{v})$ satisfies $\langle \vec{u}, \vec{x} \rangle = 0$. Now we claim that

$$\Pr[G_{m1}(A)] = \Pr[G_{m0}(A)] \tag{2.19}$$

Let us justify this. If the “If” statement at line 14 is true, we have, by the above, $\langle \vec{t} - \vec{t}', \vec{x} \rangle = 0$, or $\langle \vec{t}, \vec{x} \rangle = \langle \vec{t}', \vec{x} \rangle$, and so, as per line 8 of Figure 2.8, ought indeed to set $TV[\vec{t}] = TV[\vec{t}']$. The inclusion of the boxed code at line 18 further ensures consistency with line 8 of Figure 2.8. So $VE$ is returning the same things in games $G_{m1}, G_{m0}$. While $DLO$ defines some new quantities, what it returns does not change compared to game $G_{m0}$. This concludes the justification of Eq. (2.19).

Games $G_{m1}, G_{m2}$ are identical-until-bad as defined in [19]. Let $B_2$ be the event that $G_{m2}(A)$ sets bad. Then by the Fundamental Lemma of Game Playing [19],

$$\Pr[G_{m1}(A)] \leq \Pr[G_{m2}(A) \text{ and } \overline{B_2}] + \Pr[B_2], \tag{2.20}$$

where $\overline{B_2}$ denotes the complement of event $B_2$. We claim that

$$\Pr[G_{m2}(A) \text{ and } \overline{B_2}] + \Pr[B_2] \leq \Pr[G_{m3}(A)], \tag{2.21}$$

where game $G_{m3}$ is in Figure 2.10. It includes the boxed code, which game $G_{m4}$ excludes. In these games, $VE$ returns the same thing as in game $G_{m2}$, but also indexes (keeps track of) vectors $\vec{t}$ that might set bad in $G_{m2}$, so that it can refer to them in $F1N$. The achievement is that this procedure no longer refers to $\vec{x}$. Now we would like the same to be true for $DLO$. A natural approach would be to have $DLO$ return a random $z \leftarrow \mathbb{Z}_p$. However, the true distribution of $z$ is more complex, and instead we will use Lemma 2.5.2. Line 11 sets $w \in \mathbb{Z}_p$ to be the $i$-th coordinate of vector $\vec{w}$. Line 12 checks if $\vec{w}$ is 0 at all but its $i$-th coordinate, if so correctly returning $w$ as the answer to the oracle query. At lines 13,14, the choices of $z$ and $x_i$ are made in
justifying Eq. (2.21).

The Lemma thus implies that in game \( G_m \), the returned \( z \) is distributed as it is in game \( G_m^2 \). \( \text{FIN} \) of game \( G_m \) returns true if either \( y = y' \), or game \( G_m^2 \) would set \( \text{bad} \), justifying Eq. (2.21).

Games \( G_m^3, G_m^4 \) are identical-until-bad, so by the Fundamental Lemma of Game Play-

\begin{verbatim}
INIT(): / G_m^3–G_m^5, G_m^\alpha,\beta.
1 p ← |G| ; 1 ← VE(]) ; g ← VE(\bar{e}_{n+1}) ; y ← VE(\bar{e}_{n+2})
2 For \( i = 1, \ldots, n \) do \( X_i ← VE(\bar{e}_i) \)
3 Return 1, g, y, X_1, \ldots, X_n

VE(\bar{t}): / G_m^3–G_m^5, G_m^\alpha,\beta. Here \( \bar{t} ∈ \mathbb{Z}_{p}^{n+2} \).
4 If (TV[\bar{t}] \neq \perp) then return TV[\bar{t}]
5 \( C ← G \setminus GL \)
6 If (\( \exists \bar{t}': (TV[\bar{t}'] \neq \perp \text{ and } \bar{t'} = \text{span}(\bar{t}) ) \)) then \( C ← TV[\bar{t}'] \)
7 Else \( k ← k + 1 ; \bar{t}_k ← \bar{t} ; GL ← GL \cup \{ C \} \)
8 TV[\bar{t}] ← C ; TI[C] ← \bar{t} ; Return TV[\bar{t}]

\( \text{VE}^{-1}(C) \): / G_m^3–G_m^5, G_m^\alpha,\beta. Here TI[C] \neq \perp.
9 Return TI[C]

\( \text{Op}(A, B, \text{sgn}) \): / G_m^3–G_m^5, G_m^\alpha,\beta. Here TI[A], TI[B] \neq \perp and \( \text{sgn} \in \{ +, - \} \)
10 \( \bar{c} ← \text{VE}^{-1}(A) \text{ sgn } \text{VE}^{-1}(B) ; C ← \text{VE}(\bar{c}) \); Return C

DLO(i, W): / [G_m^3, G_m^4]. Here \( i ∈ [n] \) and TI[W] \neq \perp.
11 \( \bar{w} ← \text{VE}^{-1}(W) ; w ← \bar{w}[i] \)
12 If (\( \bar{w} = 0 \)) then return \( w \)
13 \( \bar{z} ← z_p \setminus \{ w \} ; y ← z_p \setminus \{ x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n \} \)
14 \( x_i ← (x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n, 1, y) ; x_i ← (z - w)^{-1} \cdot (\bar{w}, x_i) \)
15 If (\( \langle \bar{w}, x_i \rangle = 0 \)) then \( \text{bad} ← \text{true} ; \bar{z} ← w ; z ← z_p \)
16 \( \bar{v} ← \bar{w} - z \cdot \bar{e}_i ; \text{Return } z \)

\( \text{FIN}(y') \): / G_m^3, G_m^4.
17 \( x ← (x_1, \ldots, x_n, 1, y) \)
18 Return \( (y = y') \text{ or } (\exists \alpha, \beta : 1 ≤ \alpha < \beta ≤ k \text{ and } \langle \bar{t}_\alpha - \bar{t}_\beta, x \rangle = 0 ) \)
\end{verbatim}

Figure 2.10: Procedures for games \( G_m^3, G_m^4 \) in the proof of Theorem 2.5.1 Some procedures, as marked, will be used in later games.

accordance with one case of Lemma 2.5.2 with \( y \), and the \( x_j \) for \( j \neq i \), chosen correctly. Line 15 checks if it is the other case that happened, and, if so, game \( G_m^3 \) corrects the choices of \( z, x_i \) according to the Lemma. The Lemma thus implies that in game \( G_m^3 \), the returned \( z \) is distributed as it is in game \( G_m^2 \). \( \text{FIN} \) of game \( G_m^3 \) returns true if either \( y = y' \), or game \( G_m^2 \) would set \( \text{bad} \), justifying Eq. (2.21).
DLO(i,W): \( \Gamma \text{Gm}_5, \text{Gm}_{\alpha,\beta} \). Here \( i \in [n] \) and \( \text{TI}(W) \neq \perp \).

DLO(i,W):
19. \( \vec{w} \leftarrow \text{VE}^{-1}(W) ; w \leftarrow \vec{w}[i] \)
20. If (\( \vec{w} - w \cdot \vec{e}_i = \vec{0} \)) then return \( w \)
21. \( z \leftarrow \mathbb{Z}_p \setminus \{ w \} \) ; \( \vec{v} \leftarrow \vec{w} - z \cdot \vec{e}_i \) ; Return \( z \)

\text{FIN}(y'):\ \( \Gamma \text{Gm}_5 \).
22. \( y \leftarrow \mathbb{Z}_p \) ; Return (\( y = y' \))

\text{FIN}(y'):\ \( \Gamma \text{Gm}_{\alpha,\beta} \).
23. If (not (1 \leq \alpha < \beta \leq k)) then return false
24. \( y \leftarrow \mathbb{Z}_p \) ; \( x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n \leftarrow \mathbb{Z}_p^* \)
25. \( \vec{x}_i \leftarrow (x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n, 1, y) \) ; \( x_i \leftarrow (z - w) \cdot (\vec{w}, \vec{x}_i) \)
26. \( \vec{x} \leftarrow (x_1, \ldots, x_n, 1, y) \)
27. Return (\( \langle \vec{t}_\alpha - \vec{t}_\beta, \vec{x} \rangle = 0 \))

**Figure 2.11:** Further procedures to define game \( \text{Gm}_5 \) and games \( \text{Gm}_{\alpha,\beta} \) (1 \leq \alpha < \beta \leq q) in the proof of Theorem 2.5.1.

\[
\text{Pr}[\text{Gm}_3(\mathcal{A})] \leq \text{Pr}[\text{Gm}_4(\mathcal{A})] + \text{Pr}[\text{Gm}_4(\mathcal{A}) \text{ sets bad}] .
\] (2.22)

We claim

\[
\text{Pr}[\text{Gm}_4(\mathcal{A}) \text{ sets bad}] \leq \frac{1}{p - 1} .
\] (2.23)

That is, the probability that \( \langle \vec{w}, \vec{x}_i \rangle = 0 \) at line 15 is at most \( 1/(p - 1) \). We now justify this. Line 12 tells us that, at line 15, there is some \( j \in [1..n+2] \setminus \{ i \} \) such that \( \vec{w}[j] \neq 0 \). Consider two cases. The first is that there is such a \( j \) satisfying \( j \neq n + 1 \). If \( j = n + 2 \), there is exactly one choice of \( y \in \mathbb{Z}_p \) making \( \langle \vec{w}, \vec{x}_i \rangle = 0 \), while if \( j \in [1..n] \setminus \{ i \} \), there is at most one choice of \( x_j \in \mathbb{Z}_p^* \) making \( \langle \vec{w}, \vec{x}_i \rangle = 0 \), so overall the probability that \( \langle \vec{w}, \vec{x}_i \rangle = 0 \) is at most \( 1/(p - 1) \). The second case is that \( \vec{w}[j] = 0 \) for all \( j \neq n + 1 \). But then the probability that \( \langle \vec{w}, \vec{x}_i \rangle = 0 \) is zero. This completes the justification of Eq. (2.23).

We now define a game \( \text{Gm}_5 \), and also a game \( \text{Gm}_{\alpha,\beta} \) for each \( 1 \leq \alpha < \beta \leq q \), where
\[ q = Q^{op}_q + n + 3. \] The DLO, Fin procedures of these games are shown in Figure 2.11 and the other procedures remain as in Figure 2.10. Since the boxed code is absent in DLO of game \( Gm_4 \), the only random choice it needs to make is \( z \), yielding the simplified DLO procedure of Figure 2.11. The other random choices are delayed to Fin. The event resulting in game \( Gm_4 \) returning true is broken up in the new games so that, by the union bound,

\[
\Pr[Gm_4(\mathcal{A})] \leq \Pr[Gm_5(\mathcal{A})] + \sum_{1 \leq \alpha < \beta \leq q} \Pr[Gm_{\alpha,\beta}(\mathcal{A})].
\] (2.24)

Clearly

\[
\Pr[Gm_5(\mathcal{A})] \leq \frac{1}{p}.
\] (2.25)

Now, fix any \( 1 \leq \alpha < \beta \leq q \). We assume wlog that \( k \) always equals \( q \). In game \( Gm_{\alpha,\beta} \), let \( \bar{d} = \bar{i}_\alpha - \bar{i}_\beta \), let \( a = (z - w)^{-1} \) and let \( \bar{u} = a \cdot \bar{d}[i] \cdot \bar{w} + \bar{d} \). Let \( Z \) be the event that \( \langle \bar{d}, \bar{x} \rangle = 0 \), and let \( S \) be the event that \( \bar{d} \in \text{span}(\bar{v}) \). Then

\[
\Pr[Gm_{\alpha,\beta}(\mathcal{A})] = \Pr[Z] = \Pr[Z \text{ and } \bar{S}] + \Pr[Z \text{ and } S]
\leq \Pr[Z \mid \bar{S}] + \Pr[S].
\] (2.26)

We will show that

\[
\Pr[Z \mid \bar{S}] \leq \frac{1}{p - 1}.
\] (2.27)

\[
\Pr[S] \leq \frac{1}{p - 1}.
\] (2.28)
We now justify Eq. (2.27). We have
\[
\langle \bar{d}, \bar{x} \rangle = \bar{x}_i \cdot \bar{d}[i] + \langle \bar{d}, \bar{x}_i \rangle = a \cdot \langle \bar{w}, \bar{x}_i \rangle \cdot \bar{d}[i] + \langle \bar{d}, \bar{x}_i \rangle = \langle a \cdot \bar{d}[i] \cdot \bar{w} + \bar{d}, \bar{x}_i \rangle = \langle \bar{u}, \bar{x}_i \rangle
\]

Assume \( \bar{d} \not\in \text{span}(\bar{v}) \), meaning event \( \bar{S} \) happens. Then we claim (we will justify this in a bit) that there exists a \( j \in [1..n+2] \setminus \{i, n+1\} \) such that \( \bar{u}[j] \neq 0 \). This means that the random choice of either \( x_j \) (if \( j \in [1..n] \setminus \{i\} \)) or \( y \) (if \( j = n+2 \)) has probability at most \( 1/(p-1) \) of making \( \langle \bar{u}, \bar{x}_i \rangle = 0 \). To justify the claim, suppose to the contrary that for all \( j \in [1..n+2] \setminus \{i, n+1\} \) we have \( \bar{u}[j] = 0 \). Since \( \langle \bar{u}, \bar{x}_i \rangle = 0 \), it must be that \( \bar{u}[n+1] = 0 \) as well. Let \( b = -a \cdot \bar{d}[i] \), so that \( \bar{d}[i] = -b \cdot a^{-1} = -b \cdot (z - w) = b \cdot (w - z) \). For \( j \in [1..n+2] \setminus \{i\} \) we have \( a \cdot \bar{d}[i] \cdot \bar{w}[j] + \bar{d}[j] = 0 \), or \( \bar{d}[j] = -a \cdot \bar{d}[i] \cdot \bar{w}[j] = b \cdot \bar{w}[j] \). Recalling that \( \bar{v} = \bar{w} - z \cdot \bar{e}_i \) and \( w = \bar{w}[i] \), we see that \( \bar{d} = b \cdot \bar{v} \), which puts \( \bar{d} \) in \( \text{span}(\bar{v}) \), contradicting our assumption that \( \bar{d} \not\in \text{span}(\bar{v}) \). This concludes the justification of Eq. (2.27).

We turn to Eq. (2.28). Suppose \( \bar{d} \in \text{span}(\bar{v}) \), meaning \( \bar{d} = b \cdot \bar{v} = b \cdot \bar{w} - b \cdot \bar{z} \cdot \bar{e}_i \) for some \( b \in \mathbb{Z}_p^* \). By line 4 of Figure 2.10, \( \bar{t}_\alpha \neq \bar{t}_\beta \), so \( \bar{d} \neq \bar{0} \) so \( b \neq 0 \). So there is at most one \( z \in \mathbb{Z}_p \) such that \( \bar{d}[i] = bw - b \cdot \bar{z} \), and our \( z \) chosen at random from \( \mathbb{Z}_p \setminus \{w\} \) has probability at most \( 1/(p-1) \) of being this one.

Putting the above together we have
\[
\text{Adv}_{G,n,m}^{\text{gg-\text{mbdl}}} (\mathcal{A}) \leq \frac{1}{p-1} + \frac{1}{p} + \frac{q(q-1)}{2} \cdot \frac{2}{p-1} = \frac{1 + q(q-1)}{p-1} + \frac{1}{p}.
\]

This concludes the proof. \( \blacksquare \)
2.6 Okamoto Identification and Signatures from MBDL

In this section, we give a tight reduction of the IMP-PA security of the Okamoto identification scheme to the 1-MBDL problem and derive a corresponding improvement for Okamoto signatures.

**Figure 2.12.** Let $G$ be a group of prime order $p = |G|$ and let $g \in G^*$ be a generator of $G$. The Okamoto ID scheme $ID = \text{OkalID}(G,g)$ is shown pictorially at the top and algorithmically at the bottom left. At the bottom right is the Okamoto signature scheme $DS = \text{OkaSig}(G,g)$, using $H : G \times \{0,1\}^* \rightarrow \mathbb{Z}_p$.

**OKAMOTO IDENTIFICATION SCHEME AND PRIOR RESULTS.** Let $G$ be a group of prime order $p = |G|$, and $g \in G^*$ a generator of $G$. We recall the Okamoto identification scheme [74] $ID = \text{OkalID}(G,g)$ in Fig. 2.12. The public key has the form $pk = (g_2, X) \in G^2$ where $g_2$ is
a generator and \( X = g^{x_1}g_2^{x_2} \), where the secret key is \( sk = (g_2, x_1, x_2) \in \mathbb{Z}_p^3 \). The commitment is \( R = g^{r_1}g_2^{r_2} \in G \), and \((r_1, r_2)\) is returned as the prover state by the commitment algorithm. Challenges are drawn from \( \text{ID.Chl} = \mathbb{Z}_p \), and the response \( z \) and decision \( b \) are computed as shown.

Given an IMP-PA adversary \( \mathcal{A} \) against \( \text{ID} = \text{OkalD}[G, g] \), the classical proof of \([74]\) builds a DL-adversary \( \mathcal{B} \), as follows. On input a target point \( Y \) whose discrete-log it wants to compute, \( \mathcal{B} \) sets \( g_2 = Y \). It then itself picks \( x_1, x_2 \) and sets \( X = g^{x_1}g_2^{x_2} \), so that \((x_1, x_2)\) is what’s called a representation of \( X \). Now \( \mathcal{B} \) runs \( \mathcal{A} \) on public key \((g_2, X)\). Knowing the secret key \((g_2, x_1, x_2)\), it is easy for \( \mathcal{B} \) to simulate the \( \text{Tr} \) oracle. When \( \mathcal{A} \) makes its impersonation attempt, rewinding is used, as usual, to obtain two accepting conversation transcripts with the same commitment \( R^* \). From these, \( \mathcal{B} \) can compute another representation of \( X \), namely some \( a_1, a_2 \) such that \( X = g^{a_1}g_2^{a_2} \). The witness indistinguishability property of the protocol says that \((a_1, a_2) \neq (x_1, x_2)\), except with probability \( 1/p \). Finally, from the two distinct representations of \( X \), adversary \( \mathcal{B} \) can compute \( \text{DL}_{G, g}(g_2) \). Again the simplest analysis is via the Reset Lemma of \([15]\), which says that

\[
\text{Adv}^{\text{imp-pa}}_{\text{ID}}(\mathcal{A}) \leq \sqrt{\text{Adv}^{\text{dl}}_{G, g}(\mathcal{B})} + \frac{2}{p}, \tag{2.29}
\]

the extra \( 1/p \) term compared to Equation (2.7) being due to the probability that the two representations are equal. The running time \( T_B \) of \( \mathcal{B} \) is roughly \( 2T_A \) plus simulation overhead \( O(Q_{\mathcal{A}}^{\text{Tr}} \cdot T_G^e) \), where \( T_G^e \) is the time for an exponentiation in \( G \).

**Our result.** We show that the IMP-PA-security of the Okamoto identification scheme reduces tightly to the 1-MBDL problem. As with Schnorr, the reduction does not use rewinding.

**Theorem 2.6.1** Let \( G \) be a group of prime order \( p = |G| \), and let \( g \in G^* \) be a generator of \( G \). Let \( \text{ID} = \text{OkalD}[G, g] \) be the Okamoto identification scheme. Let \( \mathcal{A} \) be an adversary attacking the imp-pa security of \( \text{ID} \). Then we can construct an adversary \( \mathcal{B} \) (shown explicitly in Figure 2.13)
such that

\[
\text{Adv}^{\text{imp-pa}}_{\text{ID}}(\mathcal{A}) \leq \text{Adv}^{\text{mbdl}}_{G,\mathcal{G},1}(\mathcal{B}) + \frac{1}{p}.
\] (2.30)

Additionally, \( T_B \) is roughly \( T_A \) plus simulation overhead \( O(Q^{\text{Tr}}_A \cdot T^e_0) \).

**Proof of Theorem 2.6.1**: Our reduction from MBDL deviates from the prior one discussed above. It does not set \( g_2 \) to the target point \( Y \), instead picking \( w \) and setting \( g_2 = g^w \). It sets \( X \) to a base under which it can take a discrete logarithm. When adversary \( \mathcal{A} \) provides \( R^* \) in its impersonation attempt, adversary \( \mathcal{B} \) picks \( c^* \) so that \( Y = R^* X^{c^*} \). Then, from \( \mathcal{A} \), it gets \((z_1, z_2)\) satisfying \( g^{z_1} g^{z_2} = R^* X^{c^*} = Y \). Using \( w \), adversary \( \mathcal{B} \) then finds \( \text{DL}_{G,\mathcal{G}}(Y) \). It simulates the Tr oracle using the zero-knowledge simulator. Thus, while in the prior approach the reduction knows the secret key but not \( \text{DL}_{G,\mathcal{G}}(g_2) \), in ours the reduction does not know the secret key but knows \( \text{DL}_{G,\mathcal{G}}(g_2) \).

For the formal proof, we claim that the adversary \( \mathcal{B} \), shown in Fig. 2.13, satisfies Equation (2.30). Since the analysis is similar to that in the proof of Theorem 2.4.1 we will be brief. The \( X \) provided by \( \mathcal{B} \) to \( \mathcal{A} \) is a generator. In the scheme, \( X = g^{x_1 + w x_2} \) fails to be generator iff \( x_1 + w x_2 = 0 \), which happens with probability \( 1/p \), accounting for this additive term in the bound. Adversary \( \mathcal{B} \) simulates the transcript oracle correctly by the usual zero-knowledge method. If \( \mathcal{A} \) succeeds, we have \( g^{z_1} g^{z_2} = R^* X^{c^*} \). But \( g^{z_1} g^{z_2} = g^{z_1 + w z_2} \) and \( R^* X^{c^*} = Y \), so \( z_1 + w z_2 \) can be
returned as the discrete log of $Y$.

**OKAMOTO SIGNATURES.** The Okamoto signature scheme $DS = OkaSig[G, g]$ is derived by applying the Fiat-Shamir transform [45] to the Okamoto identification scheme. Its algorithms are shown at the bottom right of Fig. 2.12. The set $DS.HF$ consists of all functions $h : G \times \{0, 1\}^* \rightarrow \mathbb{Z}_p$.

Combining Lemma 2.4.2 with Theorem 2.6.1, we get the following reduction, of the UF security of the Okamoto signature scheme to the 1-MBDL problem, that loses only a factor of the number of hash-oracle queries of the adversary.

**Theorem 2.6.2** Let $G$ be a group of prime order $p = |G|$, and let $g \in G^*$ be a generator of $G$. Let $DS = OkaSig[G, g]$ be the Okamoto signature scheme. Let $A$ be an adversary attacking the uf security of $ID$. Let $\beta = (1 + Q^H_A + Q^\text{SIGN}_A)Q^\text{SIGN}_A + (1 + Q^H_A)$. Then we can construct an adversary $B$ such that

$$ Adv^\text{uf}_{DS}(A) \leq (1 + Q^H_A) \cdot Adv^\text{mbdl}_{G, g, 1}(B) + \frac{\beta}{p}. $$

(2.31)

Additionally, $T_B$ is roughly $T_A$ plus simulation overhead $O(Q^\text{SIGN}_A \cdot T_e^G)$.

As before, the best prior result, obtained via the general Forking Lemma of [14], said that given an adversary $A$ attacking the UF security of $DS$, one can construct a discrete log adversary $B$ such that

$$ Adv^\text{uf}_{DS}(A) \leq \sqrt{(1 + Q^H_A) \cdot Adv^\text{dl}_{G, g}(B) + \frac{\beta}{p}}, $$

(2.32)

where $\beta$ and $T_B$ are as above. Roughly the bound in Eq. (2.31) is the square of the one in Eq. (2.32), and thus (always) smaller.

### 2.7 Ratio-based tightness

KMP [58] claims a tight reduction between passive impersonation security of Schnorr identification and discrete log. Their results are claimed to be tight when evaluated under
time-to-success ratio. We show here why their result does not give bounds that are as good as ours.

Let ID be the Schnorr identification scheme defined in Section 2.4. Let A be an adversary against the IMP-PA security of ID with running time $T_A$. For any given parameter $N \geq 1$, KMP [58] (Lemma 3.5) construct a DL adversary $D_N$ such that

$$\sqrt{\text{Adv}^\text{dl}_{G,g}(D_N)} \geq 1 - \left[ 1 - \left( \text{Adv}^\text{imp-pa}_{ID}(A) - \frac{1}{p} \right) \right]^N,$$

(2.33)

and $T_{D_N} = 2N \cdot T_A$. Notice that when $N = 1$, this is identical to Eq. (2.7), meaning there is no improvement in that case. Next, KMP [58] pick a specific value of $N$ that we call $N^*$. This value is $N^* = (\text{Adv}^\text{imp-pa}_{ID}(A) - 1/p)^{-1}$. So the term on the right hand side of Eq. (2.33) becomes

$$1 - \left[ 1 - \left( \text{Adv}^\text{imp-pa}_{ID}(A) - \frac{1}{p} \right) \right]^{N^*} \approx 1 - \frac{1}{e} \approx 0.63,$$

(2.34)

a constant close to 1. Let $B^* = D_{N^*}$ be the DL adversary for this parameter choice. Then, neglecting $1/p$ as being essentially 0, one has

$$\text{Adv}^\text{dl}_{G,g}(B^*) \geq \left( 1 - \frac{1}{e} \right)^2 \approx 0.4$$

(2.35)

$$T_{B^*} = 2N^* \cdot T_A \approx \frac{T_A}{\text{Adv}^\text{imp-pa}_{ID}(A)}.$$

(2.36)

Dividing, they obtain the ratio tightness

$$\frac{\text{Adv}^\text{imp-pa}_{ID}(A)}{T_A} \leq \frac{\text{Adv}^\text{dl}_{G,g}(B^*)}{T_{B^*}}.$$

(2.37)

“Tightness” is claimed because the time-to-success ratio is preserved. However, we will show that one cannot use the above to instantiate parameters that as competitive as the ones guaranteed by our bounds. This is because the running time $T_{B^*}$ from Eq. (2.36) is in general much larger than
and the ratio tightness only holds when the running time of the DL adversary is increased in this way to make its advantage a constant as per Eq. (2.35).

As before, let us the GGM bound for breaking DL, i.e. $Adv_{G,G}^{dl} (\mathcal{B}^*) \leq T_{\mathcal{B}^*}^2 / p$. Then, from Eq. (2.35) one has $T_{\mathcal{B}^*} \approx \sqrt{0.4 \cdot p}$, so

$$\frac{Adv_{ID}^{imp-pa} (\mathcal{A})}{T_{\mathcal{A}}} \leq \frac{0.4}{\sqrt{0.4 \cdot p}},$$

which means that one would need a group of size

$$p \approx \left( \frac{T_{\mathcal{A}}}{Adv_{ID}^{imp-pa} (\mathcal{A})} \right)^2.$$  

This is exactly the same requirement as dictated by the prior results, namely Equation (2.7) and Equation (2.11). Hence, the guarantee by the results of KMP is the same as offered by prior results in Fig. 2.1.

2.8 Acknowledgements

We thank the Indocrypt 2020 reviewers for their insightful comments.

Bellare was supported in part by NSF grant CNS-1717640 and a gift from Microsoft. Dai was supported in part by a Powell Fellowship and grants of the Bellare.

Chapter 2, in full, is a reprint of the material as it appears in International Conference on Cryptology in India, 2020. Bellare, Mihir; Dai, Wei. The dissertation author was the primary investigator and author of this paper.
Chapter 3

Chain Reductions for Multi-signatures and the HBMS Scheme

3.1 Introduction

Usage in cryptocurrencies has lead to interest in practical, Discrete-Log-based multi-signature schemes. Proposals exist, are efficient, and are supported by proofs, BUT, the bound on adversary advantage in the proofs is so loose that the proofs fail to support use of the schemes in the 256-bit groups in which they are implemented in practice. This leaves the security of in-practice schemes unclear.

We ask, is it possible to bridge this gap to give some valuable support, in the form of tight reductions, for in-practice schemes? As long as we stay in the current paradigm, namely standard-model proofs from DL, the answer is likely NO. To make progress, we need to be willing to change either the model or the assumption. We show that in fact changing either suffices. Our approach is to give, for any scheme, many different paths to security. In particular we give (1) tight reductions from DL in the Algebraic Group Model (AGM) [47], and (2) tight, standard-model reductions from well-founded assumptions other than DL. We obtain these results
via a framework in which a reduction is “factored” into a chain of sub-reductions involving intermediate problems.

We implement this approach first with classical 3-round schemes, giving chain reductions yielding (1) and (2) above for the BN [14] and MuSig [64] schemes. Then, in the space of 2-round schemes, we give a new, efficient scheme, called HBMS, for which we do the same. We now look at all this in more detail.

**Background.** A multi-signature $\sigma$ on a message $m$ can be thought of as affirming that “We, the members of this group, all, jointly, endorse $m$.” The group is indicated by the vector $\mathbf{vk} = (\mathbf{vk}[1], \ldots, \mathbf{vk}[n])$ of individual public verification keys of its members, and can be dynamic, changing from one signature to another. Signing is done via an interactive protocol between group members; each member $i$ begins with its own public verification key $\mathbf{vk}[i]$, its matching private signing key $\mathbf{sk}[i]$, and the message $m$, and, at the end of the interaction, they output the multi-signature $\sigma$. The latter should be compact (of size independent of the size of the group), precluding the trivial solution in which $\sigma$ is a list of the individual signatures of the group members on $m$.

Following its suggestion in the 1980s [56], the primitive has seen much evolution [52, 60, 72, 66, 14]. Early schemes assumed all signers in the signing protocol picked their verification keys honestly. “Rogue-key attacks,” in which a malicious signer picked its verification key as function of that of an honest signer, lead to an upgraded target, schemes that retain security even in the presence of adversarially-chosen verification keys. Towards this challenging end we first saw schemes either using interactive key-generation [66] or making the “knowledge of secret key” assumption [21, 61]. Finally, BN [14] gave an efficient, Schnorr-based scheme in the “plain public-key” model, where security was provided even in the face of maliciously-chosen verification keys, yet no more was assumed about these keys than their having certificates as per a standard PKI.

The BN model and definition have become the preferred target; it is the one used in the
schemes we discuss next, and in our scheme as well. We denote the security goal as MS-UF. In Section 3.4 we define it via a game, and define the ms-uf advantage of an adversary as its probability of winning this game.

A NEW WAVE. Applications in blockchains and cryptocurrencies —see [25] for details— have fueled a resurgence of interest in multi-signatures. The desire here is MS-UF-secure, DL-based schemes that work over standard elliptic curves such as Secp256k1 or Curve25519. (Pairing-based schemes [25] are thus precluded.) The natural candidate is BN. But the new application arena has lead to a desire for the following further features, not possessed by BN: (1) Key aggregation. There should be a way to aggregate a set of verification keys into a single, short aggregate key, relative to which signatures are verified. (2) Two rounds. A signing protocol using only 2 rounds of interaction, as opposed to the 3 used by BN.

MuSig [64, 25] broke ground by adapting BN to add key aggregation. Now the effort moved to reducing the number of rounds. This proved challenging. Early proposals of two-round schemes —[8, 62, 85] as well as an early, two-round version of MuSig [64]— were broken by DEFKLNS [40]. To fill the gap, DEFKLNS gave a new two-round scheme, mBCJ. Other proposals followed: MuSig2 [69], MuSig-DN [70] and DWMS [4]. All these support key aggregation.

All the schemes discussed here come with proofs of MS-UF security based on the hardness of the DL (Discrete Log) problem in the underlying group $G$, up to variations in the model (standard or AGM [47]) or the type of DL problem (plain or OMDL [13]).

CURRENT BOUNDS. On being informed that a scheme has a proof of security based on the hardness of the DL problem in an underlying elliptic-curve group $G$, the expectation of a practitioner is that the probability that a time $t$ attacker can violate MS-UF security is no more than the probability of successfully computing a discrete logarithm in $G$, which, as per [81], is $t^2/p$, where $p$, a prime, is the size of $G$. Concretely, with the 256-bit curves Secp256k1 or Curve25519 —$p \approx 2^{256}$— they would expect that a time $t \approx 2^{80}$ attacker has ms-uf advantage at
Table 3.1: Bounds on ms-uf advantage for the 3-round schemes BN and MuSig. First we show prior bounds, then ours. In each case we first show the upper bound $UB_{MS}^{ms-uf}(t,q,q_s,p)$ as a formula, where $t,q,q_s$ are, respectively the adversary running time, the number of its RO queries and the number of executions of the signing protocol, while prime $p$ is the size of the underlying group $G$. We then show the evaluation with $t = q = 2^{80}$, $q_s = 2^{30}$ and $p \approx 2^{256}$, to capture security over 256-bit curves Secp256k1 or Curve25519. Our bounds assume generic hardness of IDL (for BN) and XIDL (for MuSig), which both hold in AGM. Our bounds are also tight and optimal, matching those for the DL problem itself.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Previous $UB_{MS}^{ms-uf}(t,q,q_s,p)$</th>
<th>Ours $UB_{MS}^{ms-uf}(t,q,q_s,p)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN [14]</td>
<td>$\sqrt{(q \cdot t^2)/p}$</td>
<td>$2^{-8}$</td>
</tr>
<tr>
<td>MuSig [25, 64]</td>
<td>$4\sqrt{(q^3 \cdot t^2)/p}$</td>
<td>$1$</td>
</tr>
</tbody>
</table>

Most $2^{160–256} = 2^{-96}$.

But this expectation is only correct if the reduction in the proof is tight. Current proofs for DL-based multi-signature schemes are loose. With the 256-bit curves Secp256k1 or Curve25519, and for a $2^{80}$-time attacker, the proof of [14] for BN can preclude only a $2^{-8}$ ms-uf advantage, while the proof of [64, 25] for MuSig cannot even preclude a ms-uf advantage of 1, meaning there may be, per the proof, no security at all (cf. Figure 3.1). For 2-round schemes, the advantage precluded by current proofs is $2^{-16}$ in one case, and again just 1 for the others (cf. Figure 3.1). Overall, the proofs fail, by big margins, to support the parameter choices and expectations of practice.

Before continuing, let us expand on the above estimates. A proof of MS-UF security for a multi-signature scheme MS gives a formula $UB_{MS}^{ms-uf}(t,q,q_s,p)$ that upper bounds the ms-uf advantage of an adversary as a function of its running time $t$, the number $q$ of its queries to the random oracle, and the number $q_s$ of executions of the signing protocol in the chosen-message attack in the ms-uf game. They are shown in Figures 3.1 and 3.1. We assume that $t \geq q \geq q_s$ to get these formulas, we first assume that the best attack against the DL problem is generic, so that a time $t$ attacker has success probability at most $t^2/p$ [81]. Next, we use the concrete-security
Figure 3.1: Bounds on ms-uf advantage for 2-round schemes. First we show bounds for prior schemes, then the bounds for our new scheme HBMS. As before, we first show the upper bound formula $UB_{MS}^{ms-uf}(t, q, q_s, p)$, where $t, q, q_s$ are, respectively the adversary running time, the number of its RO queries and the number of executions of the signing protocol, while prime $p$ is the size of the underlying group $G$. We then show the evaluation with $t = q = 2^{80}$, $q_s = 2^{30}$ and $p \approx 2^{256}$, to capture security over 256-bit curves Secp256k1 or Curve25519. For MuSig2, results differ depending on a parameter $\nu$ of the scheme. We also show estimates of signing time (per signer) and verification time. Here $T_{me}^n$ is the time to compute one $n$-multi-exponentiation in $G$. The “NIZK” for MuSig-DN indicates that signing requires computation and verification of a NIZKs, which is (much) more expensive then other operations shown.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Security</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>mBCJ [40]</td>
<td>$(q_3 \cdot q^2 \cdot t^2)/p$</td>
<td>Sign: $T_{me}^2 + T_{me}^3$</td>
</tr>
<tr>
<td>MuSig-DN [70]</td>
<td>$\sqrt{(q_3 \cdot t^2)/p}$</td>
<td>Vf: $3T_{me}^n$</td>
</tr>
<tr>
<td>MuSig2, $\nu \geq 4$ [69]</td>
<td>$\sqrt{(q^3 \cdot t^2)/p}$</td>
<td>NIZK: $T_{me}^2$</td>
</tr>
<tr>
<td>MuSig2, $\nu = 2$ [69]</td>
<td>$(t^2 + q^3)/p$</td>
<td>$T_{me}^{2n}$</td>
</tr>
<tr>
<td>DWMS [4]</td>
<td>$t^2/p + q/\sqrt{p}$</td>
<td>$T_{me}^2 + T_{me}^2/2^n$</td>
</tr>
<tr>
<td>HBMS</td>
<td>$t^2/p$</td>
<td>$T_{me}^3$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Security</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>$UB_{MS}^{ms-uf}(t, q, q_s, p)$</td>
<td>$p \approx 2^{256}$</td>
</tr>
</tbody>
</table>

results, in theorems in the papers, that give reductions from the DL problem to the MS-UF security of their scheme. The square-roots in the formulas arise from uses of forking lemmas [76, 14, 8]; the fourth-roots from nested use. The bounds in our Figures are approximate, dropping negligible additive terms. The proofs on which the bounds of Figures 3.1 and 3.1 are based, are, for BN [14], MuSig [25, 64], mBCJ [40], MuSig-DN [70], and MuSig2 ($\nu \geq 4$) [69], in the standard model; and for MuSig2 ($\nu = 2$) [69], DWMS [4] and HBMS, in the AGM. See Section 3.8 for details.

Towards better bounds. Our thesis is that proofs should provide, not merely a qualitative guarantee, but one whose bounds quantitatively support parameter choices made in practice and the indications of cryptanalysis. Accordingly we want multi-signature schemes for which we can prove tight bounds on ms-uf advantage. How are we to reach this end? Impossibility results for Schnorr signatures [75, 58], on which the multi-signature schemes under consideration are based, indicate that a search for tight reductions that are both (1) in the standard model, and
(2) from DL, is unlikely to succeed. We need to be flexible, and relax either (1) or (2). In fact we show that relaxing either suffices: We give (1) tight reductions from DL in the Algebraic Group Model (AGM) \[47\], and (2) tight, standard-model reductions from assumptions other than DL. Together, these provide valuable theoretical support for the use of practical multi-signature schemes in 256-bit groups.

**AGM.** The AGM considers a limited, but still large class of adversaries, called algebraic. When such an adversary queries a group element to an oracle, it provides also its representation in terms of prior group elements that the adversary has seen. Intuitively, the assumption is that the adversary “knows” how group elements it creates are represented. For elliptic curve groups, this appears to be a realistic assumption, and here the AGM captures natural and currently-known attack strategies.

When considering the merits of the AGM, an important one to keep in mind is that a proof in the AGM immediately implies a proof in the well-accepted Generic Group Model (GGM) of \[81\]. (So the AGM is only “better” than the GGM.) In more detail, a tight AGM reduction from DL to some problem X immediately yields a GGM bound on adversary advantage, for X, that matches the GGM bound for DL \[47\]. Thus, overall, tight AGM reductions provide a valuable guarantee. This is recognized by Fuchsbauer, Plouviez and Seurin \[48\] who use the AGM to give a tight reduction from DL to the UF security of the Schnorr signature scheme. Their result gives hope, realized here, that such reductions are possible for multi-signatures as well.

**Chain reductions.** We achieve the above ends, and more, as follows. For each multi-signature scheme MS we consider, we give a chain of reductions, starting from DL, that we depict as

\[
\text{DL} = P_0 \rightarrow P_1 \rightarrow \cdots \rightarrow P_{m-1} \rightarrow P_m = \text{MS},
\]

where \(P_1, \ldots, P_{m-1}\) are intermediate computational problems. We refer to \(m \geq 1\) as the length of the chain. For each step \(P_{i-1} \rightarrow P_i\) we provide one of the following.
1. A tight, standard-model reduction. This is the ideal and done for as many steps as possible.

2. When 1. is not possible, we give BOTH of the following:
   2.1 A tight AGM reduction, AND ALSO
   2.2 A non-tight standard-model reduction.

Since a tight standard-model reduction implies a tight AGM one, this yields a tight AGM reduction
from DL to MS, the first of our goals stated above. (A bit better, since some sub-reductions are
standard-model.) For $i$ such that the chain $P_i \rightarrow \cdots \rightarrow MS$ consists only of tight standard-model
reductions, we have a tight, standard model proof of MS from assumption $P_i$, realizing our second
goal, stated above, of tight standard-model reductions from assumptions other than DL. (Of
course how interesting or valuable this is depends on the choice of $P_i$, but as discussed below, we
are able to make well-founded choices.)

Finally, something not yet mentioned, that follows from 1 and 2.2 of the chain reductions,
is that we always have a standard model (even if non-tight) reduction $DL \rightarrow MS$. This means
that, while adding tight AGM reductions that are valuable in practice, we are not lowering the
theoretical or qualitative guarantees, these remaining as one would expect or desire.

Chain reductions can be seen as a way to implement a modular proof framework in the
style of [58], in which steps are reused across proofs for different schemes.

NEW BOUNDS FOR CLASSICAL SCHEMES. We start by revisiting the classical 3-round
schemes, namely BN and MuSig. Figure 3.2 illustrates our chains, that we now discuss.

IDL, formulated in [58] —they call it IDLOG, which we have abbreviated— is a purely
group-based problem that is equivalent to the security against parallel impersonation under key-
only attack (PIMP-KOA) of the Schnorr ID scheme. A tight GGM bound for IDL was shown by
[58], but an AGM reduction $DL \rightarrow IDL$ does not seem to be in the literature; we fill this gap by
providing it in Theorem 3.3.1. A (non-tight) standard model $DL \rightarrow IDL$ reduction is in [58], but
we slightly improve it in Theorem 3.3.2.

Now our chain for BN is $DL \rightarrow IDL \rightarrow BN$. This chain has length 2. Our main result for
BN is Theorem 3.5.1, which shows IDL $\rightarrow$ BN with a tight, standard model reduction. Putting this together with our above-mentioned tight DL $\rightarrow$ IDL AGM-reduction of Theorem 3.3.1, we get a tight DL $\rightarrow$ BN AGM-reduction. Also our tight, standard-model IDL $\rightarrow$ BN reduction says that BN is as secure as the Schnorr identification scheme, which is valuable in its own right since the latter has withstood cryptanalysis for many years.

We introduce an intermediate, purely group-based problem we call XIDL. We show IDL $\rightarrow$ XIDL with a tight AGM reduction (Theorem 3.3.3) and a (non-tight) standard-model reduction (Theorem 3.3.4).

Our chain for MuSig is DL $\rightarrow$ IDL $\rightarrow$ XIDL $\rightarrow$ MuSig. This chain has length 3. Our main result for MuSig is Theorem 3.6.1, which shows XIDL $\rightarrow$ MuSig with a tight, standard model reduction. Putting this together with the rest of the chain, we get a tight DL $\rightarrow$ MuSig AGM-reduction. If we are willing to view XIDL as an assumption extending IDL, we can also view MuSig as based tightly on that.
This means we show that $UB_{MS}^{ms-uf}(t,q,q_s,p) \leq t^2/p$ for both schemes, matching the DL bound. This is tight and optimal, since the multi-signature schemes can be broken by taking discrete-logs. Figure 3.1 compares our results with the prior ones.

**NEW 2-ROUND SCHEME.** Turning to 2-round schemes, we give a new scheme, called HBMS. HBMS supports key aggregation, in line with other 2-round schemes. Our chain for our new 2-round HBMS scheme is DL $\rightarrow$ IDL $\rightarrow$ XIDL $\rightarrow$ HBMS. This chain has length 3. We show XIDL $\rightarrow$ HBMS with a tight AGM reduction (Theorem 3.7.1) and a (non-tight) standard-model reduction (Theorem 3.7.2). Putting this together with the rest of the chain, we get a tight DL $\rightarrow$ HBMS AGM-reduction, in particular showing $UB_{MS}^{ms-uf}(t,q,q_s,p) \leq t^2/p$, matching the DL bound. We also get a (non-tight) DL $\rightarrow$ HBMS standard-model-reduction.

Figure 3.1 compares HBMS with prior 2-round schemes. It shows that our improvement in security is not at the cost of efficiency. (Signing in HBMS is as efficient, or more so, than in prior schemes. For verification, MuSig-DN [70] is slightly faster, but signing in the latter is prohibitive due to the use of NIZKs.)

As the above shows, we reuse steps across different chains. Thus XIDL is an intermediate point for both MuSig and HBMS, and IDL for both BN and XIDL. This simplifies proofs and reduces effort. It also shows common elements and relations across schemes.

**EQUIVALENCES.** As discussed above, Theorem 3.5.1 shows IDL $\rightarrow$ BN with a tight, standard model reduction. We also give, in Theorem 3.5.2, a converse, namely a tight, standard-model reduction showing BN $\rightarrow$ IDL. This shows that IDL and BN are, security-wise, equivalent. Similarly, as discussed above, Theorem 3.6.1 shows MuSig $\rightarrow$ XIDL with a tight, standard model reduction, and we also give, in Theorem 3.6.2, a converse, namely a tight, standard-model reduction showing XIDL $\rightarrow$ MuSig. This shows that XIDL and MuSig are equivalent. Overall, this shows that IDL and XIDL are not arbitrary choices, but characterizations of the schemes whose consideration is necessary.

**DEFINITIONAL CONTRIBUTIONS.** DEFKLNS [40] found subtle gaps in some prior
proofs of security for some two-round multi-signature schemes \cite{8, 62, 85}. This indicates a need for greater care in the domain of multi-signatures. We suggest that this needs to begin with definitions. The ones in prior work, stemming mostly from \cite{14}, suffer from some lack of detail and precision. In particular, the very \textit{syntax} of a multi-signature scheme is not specified in detail. This results in scheme descriptions that lack in precision, and proofs that stay at a high level in part due to lack of technical language in which to give details. This in turn can lead to bugs.

To address these issues, we revisit the definitions. We start by giving a detailed syntax that formalizes the signing protocol as a stateful algorithm, run separately by each player. Details addressed include that a player knows its position in the signer list, that player identities are separate from public keys, and integration of the ROM through a parameter describing the type of ideal hash functions needed. Then we give a security definition written via a code-based game. See Section 3.4.

**RELATED WORK.** The interest for blockchains and cryptocurrencies, and thus our focus, is DL-based schemes over elliptic curves. There are many other multi-signature schemes, based on other hard problems. Aggregate signatures \cite{27, 12} yield multi-signatures, but these use pairings (bilinear maps). A pairing-based multi-signature scheme is also given in \cite{25}. Lattice-based multi-signature schemes include \cite{42, 36}.

As noted above, IDL \cite{58} captures the security against parallel impersonation under key-only attack (PIMP-KOA) of the Schnorr ID scheme and thus, given the ZK property of the scheme, also its security against parallel impersonation under passive attack (PIMP-PA). “Parallel” means multiple impersonation attempts are allowed. IMP-PA, traditional security against impersonation under passive attack, is the case where just one impersonation attempt is allowed. The Reset Lemma \cite{15} gives a standard model DL $\rightarrow$ IMP-PA reduction. This uses rewinding and is non-tight, with a square-root loss. BD \cite{10} introduce the Multi-Base Discrete Logarithm (MBDL) problem, give a tight standard-model MBDL $\rightarrow$ IMP-PA reduction, and show that, in the GGM, the security of MBDL is the same as that of DL. An interesting open
question is whether MBDL can be used as a starting point for tight reductions for multi-signature schemes. Rotem and Segev [78] give a standard model DL → IMP-PA reduction that improves the square-root-loss reduction but is still not tight.

3.2 Preliminaries

**Notation.** If $n$ is a positive integer, then $\mathbb{Z}_n$ denotes the set $\{0, \ldots, n - 1\}$ and $[n]$ or $[1..n]$ denote the set $\{1, \ldots, n\}$. If $x$ is a vector then $|x|$ is its length (the number of its coordinates), $x[i]$ is its $i$-th coordinate and $[x] = \{x[i] : 1 \leq i \leq |x|\}$ is the set of all its coordinates. A string is identified with a vector over $\{0, 1\}$, so that if $x$ is a string then $x[i]$ is its $i$-th bit and $|x|$ is its length. By $\varepsilon$ we denote the empty vector or string. The size of a set $S$ is denoted $|S|$.

Let $S$ be a finite set. We let $x \leftarrow S$ denote sampling an element uniformly at random from $S$ and assigning it to $x$. We let $y \leftarrow A^{O_1,\ldots}(x_1,\ldots;\rho)$ denote executing algorithm $A$ on inputs $x_1,\ldots$ and coins $\rho$ with access to oracles $O_1,\ldots$, and letting $y$ be the result. We let $\rho \leftarrow \text{rand}(A)$ denote sampling random coins for algorithm $A$ and assigning it to variable $\rho$. We let $y \leftarrow A^{O_1,\ldots}(x_1,\ldots)$ be the result of $\rho \leftarrow \text{rand}(A)$ followed by $y \leftarrow A^{O_1,\ldots}(x_1,\ldots;\rho)$. We let $[A^{O_1,\ldots}(x_1,\ldots)]$ denote the set of all possible outputs of $A$ when invoked with inputs $x_1,\ldots$ and oracles $O_1,\ldots$. Algorithms are randomized unless otherwise indicated. Running time is worst case.

**Games.** We use the code-based game playing framework of [19]. (See Fig. 3.3 for an example.) Games have procedures, also called oracles. Amongst these are $\text{Init}$ and a $\text{Fin}$. In executing an adversary $\mathcal{A}$ with a game $G_m$, procedure $\text{Init}$ is executed first, and what it returns is the input to $\mathcal{A}$. The latter may now call all game procedures except $\text{Init},\text{Fin}$. When the adversary terminates, its output is viewed as the input to $\text{Fin}$, and what the latter returns is the game output. By $G_m(\mathcal{A}) \Rightarrow y$ we denote the event that the execution of game $G_m$ with adversary $\mathcal{A}$ results in output $y$. We write $\Pr[G_m(\mathcal{A})]$ as shorthand for $\Pr[G_m(\mathcal{A}) \Rightarrow \text{true}]$, the probability that the game returns true. In writing game or adversary pseudocode, it is assumed that boolean
variables are initialized to false, integer variables are initialized to 0 and set-valued variables are initialized to the empty set $\emptyset$.

A procedure (oracle) with a certain name $O$ may appear in several games. (For example, $CH$ appears in two games in Figure 3.3.) To disambiguate, we may write $G_m.O$ for the one in game $G_m$.

When adversary $A$ is executed with game $G_m$, we consider the running time of $A$ as the running time of the execution of $G_m(A)$, which includes the time taken by game procedures. By $Q_A^O$ we denote the number of queries made by $A$ to oracle $O$ in the execution. These counts are both worst case.

**GROUPS.** Throughout, $\mathbb{G}$ is a group whose order, assumed prime, we denote by $p$. We will use multiplicative notation for the group operation, and we let $1_\mathbb{G}$ denote the identity element of $\mathbb{G}$. We let $\mathbb{G}^* = \mathbb{G} \setminus \{1_\mathbb{G}\}$ denote the set of non-identity elements, which is the set of generators of $\mathbb{G}$ since the latter has prime order. If $g \in \mathbb{G}^*$ is a generator and $X \in \mathbb{G}$, then $DL_{\mathbb{G},g}(X) \in \mathbb{Z}_p$ denotes the discrete logarithm of $X$ in base $g$.

**ALGEBRAIC ALGORITHMS.** We recall the definition of algebraic algorithms [47]. As above, fix a group $\mathbb{G}$ of prime order $p$, and let $g$ be a generator. In all of our security games involving $\mathbb{G}$ and $g$, we assume that any inputs and outputs of game oracles that are group elements (meaning, in $\mathbb{G}$) are distinguished. In particular, it will be clear from the game pseudocode definition which components of inputs and outputs are such group elements. We say that an adversary, against game $G_m$, is algebraic, if, whenever it submits a group element $Y \in \mathbb{G}$ as an oracle query, it also provides, alongside, a representation of $Y$ in terms of group elements previously returned by the game oracles (the latter including $\text{INIT}$). Specifically, suppose during an execution of adversary $A$ with game $G_m$, the adversary submits a group element $Y \in G$ to game oracle $O$. Then, alongside, it must provide a vector $(v_0, v_1, \ldots, v_m) \in \mathbb{Z}_p^m$, called a representation of $Y$, such that $Y = g^{v_0} \cdot h_1^{v_1} \cdots h_m^{v_m}$, where $h_1, \ldots, h_m$ are the group elements that have been returned to the adversary by game oracles of $G_m$, so far. When considering an execution of game $G_m$ with
an adversary $\mathcal{A}$ that is not algebraic, we omit the writing of representations in the oracle calls.

**HEDGING.** Not all attacks are algebraic. The thesis of [47] is that natural ones are, and thus proving security relative to algebraic adversaries gives meaningful guarantees in practice. We adopt this here but add hedging. Recall this means that, for the same scheme, we seek both (1) A tight AGM reduction from DL, and (2) a standard-model (even if non-tight) reduction from DL. The former is used to guide and support parameter choices. The latter is viewed as at least qualitatively ruling out non-algebraic attacks.

**REDUCTIONS.** All our standard-model reductions are black-box and preserve algebraic-ness of adversaries, meaning, if the starting adversary is algebraic, so is the constructed one. This means that we can chain standard-model reductions with AGM-reductions to get overall AGM reductions.

### 3.3 Hardness of problems in groups

Our chain reductions exploit three computational problems related to groups: standard discrete log (DL); IDL [58]; and a new problem XIDL that we introduce. Here we give the definitions. We then show the length-2 chain $\text{DL} \rightarrow \text{IDL} \rightarrow \text{XIDL}$. We give reductions that are tight in the AGM and also give (non-tight) standard-model reductions, a total of four results. Referring to Figure 3.2, we are establishing the four theorems, shown in the table, that correspond to arrows 1 and 3. For the rest of the section, we fix a group $\mathbb{G}$ of prime order $p$, and a generator $g \in \mathbb{G}$.

**DL.** We recall the standard discrete logarithm (DL) problem via game $G_{\mathbb{G},g}^{\text{dl}}$ in Figure 3.3. $\text{Init}$ provides the adversary, as input, a random challenge group element $X$, and to win it must output $x' = \text{DL}_{\mathbb{G},g}(X)$ to $\text{Fin}$. We let $\text{Adv}_{\mathbb{G},g}^{\text{dl}}(\mathcal{A}) = \Pr[G_{\mathbb{G},g}^{\text{dl}}(\mathcal{A})]$ be the discrete-log advantage of adversary $\mathcal{A}$.

**IDL.** The identification discrete logarithm (IDL) problem, introduced by KMP [58].
characterizes the hardness of parallel impersonation under key-only attack (PIMP-KOA) security\footnote{KMP\cite{KMP} study IDL in the Generic Group Model (GGM)\cite{AGM} and prove a bound matching that for DL. Here, we strengthen this to give a tight AGM reduction DL \rightarrow IDL. This could be seen as implicit in part of the AGM proof of security for the Schnorr signature scheme given in\cite{AGM}, although they make no connection to IDL.} of the Schnorr identification scheme\cite{Schnorr}. Formally, consider the game $G_{m, g, q}$ given in Fig. 3.3, where parameter $q$ is a positive integer. The IDL-adversary receives a random target point $X$ from $\text{Init}$. It is additionally given access to a challenge oracle $\text{Ch}$ that can be called at most $q$ times. The oracle takes as query a group element $R$ (representing the commitment sent by the prover in Schnorr identification), stores it as $R_i$, and responds with a random challenge $c_i \leftarrow \mathbb{Z}_p$ (representing the one sent by the verifier). The adversary wins if it can produce the discrete log $z$ (representing the final prover response) of the group element $R_i \cdot X^{c_i}$, for a choice of $i$, denoted $I$, made by the adversary. We define the IDL-advantage of $\mathcal{A}$ to be

$$\text{Adv}_{G_{m, g, q}}(\mathcal{A}) = \Pr[G_{m, g, q}(\mathcal{A})].$$

**Theorem 3.3.1 [DL \rightarrow IDL, AGM]** Let $\mathbb{G}$ be a group of prime order $p$ with generator $g$. Let $q$ be a positive integer. Let $\mathcal{A}_{\text{idl}}^{\text{alg}}$ be an algebraic adversary against $G_{m, g, q}^{\text{idl}}$. Then, adversary $\mathcal{A}_{\text{dl}}$ can be constructed so that

$$\text{Adv}_{G_{m, g, q}}(\mathcal{A}_{\text{idl}}^{\text{alg}}) \leq \text{Adv}_{G_{m, g}}(\mathcal{A}_{\text{dl}}) + \frac{q}{p}.$$

Furthermore, the running time of $\mathcal{A}_{\text{dl}}$ is about that of $\mathcal{A}_{\text{idl}}^{\text{alg}}$.

The idea of the proof is as follows. Since $\mathcal{A}_{\text{idl}}^{\text{alg}}$ is algebraic, its query $R$ to $\text{Ch}$ is accompanied by $(r_1, r_2)$ such that $R = g^{r_1}X^{r_2}$. Our adversary $\mathcal{A}_{\text{dl}}$, who is running $\mathcal{A}_{\text{idl}}^{\text{alg}}$, records these as $R_i, r_{i,1}, r_{i,2}$, and responds with a random $c_i$. Eventually, $\mathcal{A}_{\text{idl}}^{\text{alg}}$ outputs $I, z$. Assuming it succeeds, we have

$$g^z = R_1 \cdot X^{c_I} = g^{r_{I,1}}X^{r_{I,2}}X^{c_I},$$

or

$$g^{z - r_{I,1}} = X^w$$

where $w = (r_{I,2} + c_I) \mod p$. Now $\text{DL}_{G_{m, g}}(X)$ can be obtained as long as $w$ has an inverse modulo $p$, meaning is non-zero. But $c_I$ was chosen at random after the adversary supplied $r_{I,2}$, so the probability that $w$ is 0 is at most $1/p$. The factor
Game $\text{Gm}_{G,g}^{dl}$

INIT:
1. $x \leftarrow Z_{|G|} ; X \leftarrow g^x$; Return $X$

FIN($x'$):
2. Return ($x = x'$)

Game $\text{Gm}_{G,g,q}^{idl}$

INIT:
1. $x \leftarrow Z_{|G|} ; X \leftarrow g^x$
2. Return $X$

CH($R$): I At most $q$ queries.
3. $i \leftarrow i + 1 ; R_i \leftarrow R$
4. $c_i \leftarrow Z_{|G|}$; Return $c_i$

FIN($I,z$):
5. Return ($g^z = R_I \cdot X^{c_i}$)

Game $\text{Gm}_{G,g,q_1,q_2}^{xidl}$

INIT:
1. $x \leftarrow Z_{|G|} ; X \leftarrow g^x$
2. Return $X$

NW$\text{TAR}(S)$: I At most $q_1$ queries.
3. $j \leftarrow j + 1 ; S_j \leftarrow S$
4. $e_j \leftarrow Z_{|G|} ; T_j \leftarrow S_j \cdot X^{e_j}$
5. Return $e_j$

CH($j\text{sel},R$): I At most $q_2$ queries.
6. $i \leftarrow i + 1 ; R_i \leftarrow R ; Y_i \leftarrow T_{j\text{sel}}$
7. $c_i \leftarrow Z_{|G|}$; Return $c_i$

FIN($I,z$):
8. Return ($g^z = R_I \cdot Y_I^{c_i}$)

Figure 3.3: Let $G$ be a group of prime order $p = |G|$, and let $g \in G^*$ be a generator of $G$. Let $q, q_1, q_2$ be positive integers. Top: Game defining discrete logarithm (DL) problem. Bottom left: Game defining identification logarithm (IDL) problem. Bottom right: Game defining random-target identification logarithm (XIDL) problem.

of $q$ accounts for the adversary’s having a choice of $I$ made after receiving challenges. The full proof is given in Section 3.10.

By $q$-IDL, we refer to IDL with parameter $q$. 1-IDL corresponds to IMP-KOA security of the Schnorr identification scheme, and a reduction $\text{DL} \rightarrow 1$-IDL is obtained via the Reset Lemma of [15]. KMP show that $1$-IDL $\rightarrow q$-IDL. Overall this gives a standard model (very non-tight) $\text{DL} \rightarrow q$-IDL reduction. However, a somewhat tighter (but still non-tight) result can be obtained when the forking lemma of [14] (which we recall as Lemma 3.9.1) is applied directly instead. Concretely, we give the following theorem, improving the prior reduction by a $\sqrt{q}$ factor. The proof is in Section 3.11.
Theorem 3.3.2 [DL → IDL, Standard Model] Let $G$ be a group of prime order $p = |G|$, and let $g \in G^*$ be a generator of $G$. Let $q$ be a positive integer. Let $A_{idl}$ be an adversary against the game $Gm_{G,g,q}^{idl}$. The proof constructs an adversary $A_{idl}$ (explicitly given in Fig. 3.11) such that

$$Adv_{G,G,q}^{idl}(A_{idl}) \leq \sqrt{q \cdot Adv_{G,G}^{idl}(A_{idl}) + \frac{q}{p}}. \tag{3.1}$$

Additionally, the running time of $A_{idl}$ is approximately $T_{A_{idl}} \approx 2 \cdot T_{A_{idl}}$.

### XIDL

We define a new problem, random target identification discrete logarithm, abbreviated XIDL. It abstracts out the algebraic core of MuSig, and we will show that its security is equivalent to the MS-UF security of MuSig. It will also be an intermediate point in our reduction chain reaching our new HBMS scheme, thereby serving multiple purposes.

With $G, p, g$ fixed as usual, XIDL is parameterized by positive integers $q_1, q_2$. Formally, consider the game $Gm_{G,g,q_1,q_2}^{xidl}$ given in Fig. 3.3. The adversary receives a randomly chosen group element $X$ from INIT. The game maintains a list $T_1, \ldots, T_{q_1}$ of “targets.” The adversary can create a target by querying the New Target oracle $NwTar$ with a group element $S$ of its choosing, whence $T_j = S \cdot X^{e_j}$ is added to the list of targets, for $e_j$ chosen randomly from $\mathbb{Z}_p$ by the game and returned to the adversary. The adversary can query the challenge oracle $Ch_{j_{sel}}(R)$ by supplying an index $j_{sel}$ and a group element $R$. The oracle records $T_{j_{sel}}$ as $Y_i$, and $R$ as $R_i$, based on the counter $i$ it maintains. Intuitively, $Ch$ is similar to the challenge oracle $Ch$ in IDL game, besides that our adversary here needs to specify the target $T_{j_{sel}}$ it is trying to impersonate against. The adversary wins the game if it can produce the discrete log $z$ of $R_{j_{seq}} \cdot Y_{I_{seq}}^{c_{I_{seq}}}$, for an index $I$ of its choice.

The oracles $NwTar$ and $Ch$ are allowed to be called at most $q_1$ and $q_2$ times, respectively. We define the XIDL advantage of $A$ as $Adv_{G,G,q_1,q_2}^{xidl}(A) = Pr[Gm_{G,G,q_1,q_2}^{xidl}(A)]$.

We show hardness of XIDL in both the AGM and the standard model, starting with the former. The theorem actually establishes the stronger $DL \to XIDL$, tightly in the AGM.
Theorem 3.3.3 [IDL $\rightarrow$ XIDL, AGM] Let $\mathbb{G}$ be a group of order $p$ with generator $g$. Let $q_1, q_2$ be positive integers. Let $A_{xidl}^{\text{alg}}$ be an algebraic adversary against $G_{\mathbb{G}, g, q_1, q_2}^{xidl}$. Then, adversary $A_{\text{dl}}$ can be constructed so that

$$\text{Adv}_{\mathbb{G}, g, q_1, q_2}^{\text{xidl}}(A_{\text{alg}}^{\text{xidl}}) \leq \text{Adv}_{\mathbb{G}, g}^{\text{dl}}(A_{\text{dl}}) + \frac{q_1 + q_2}{p}.$$ 

Furthermore, the running time of $A_{\text{dl}}$ is about that of $A_{xidl}^{\text{alg}}$.

The full proof is given in Section 3.12. Here we sketch the intuition. Since $A_{xidl}^{\text{alg}}$ is algebraic, the $j$-th query to NW TAR is of the form $S_j, s_{j,1}, s_{j,2}$ such that $S_j = g^{s_{j,1}}X^{s_{j,2}}$, and the $i$-th query to CH is of the form $j_{\text{sel}}, R_i, r_{i,1}, r_{i,2}$ such that $R_i = g^{r_{i,1}}X^{r_{i,2}}$. Let $e_j, c_i$ denote, respectively, the responses to the $j$-th query to NW TAR and the $i$-th query to CH. Eventually, $A_{xidl}^{\text{xidl}}$ outputs $I, z$.

Assuming it succeeds, the equation $g^z = R_I \cdot T_I^{c_I} = R_I \cdot (S_J \cdot X^{e_J})^{c_I}$ must hold, where $J$ was the selected index $j_{\text{sel}}$ in the $I$-th query to CH. This means that $g^z = g^{r_{I,1}}X^{r_{I,2}}(g^{s_{J,1}}X^{s_{J,2}}X^{e_J})^{c_I}$, whence $g^{z-r_{I,1}-s_{J,1}c_I} = X^w$, where $w = r_{I,2} + (s_{J,2} + e_J)c_I$. As long as $w$ is non-zero modulo $p$, one can solve for the value of $\text{DL}_{\mathbb{G}, g}(X)$. But $e_J$ and $c_I$ were independently chosen after the adversary supplied $s_{J,2}$ and $r_{I,2}$, respectively. The probability that there exists $j$ such that $(s_{J,2} + e_J) = 0$ mod $p$ is at most $q_1/p$ over $q_1$ queries to NW TAR. Assuming there is no such $j$, the probability that $w = 0$ is at most $q_2/p$, due to the $q_2$ queries to CH that $A_{xidl}^{\text{alg}}$ can make.

In the standard model, techniques in the security proof of MuSig [25] [64] could be used to show DL $\rightarrow$ XIDL, which involves two applications of the Forking Lemma, leading to a fourth-root in the bound. Instead, we give a modular result showing IDL $\rightarrow$ XIDL, using a single application of the forking lemma. The same quantitative standard model bound (with fourth-root loss) can be obtained by composing Theorem 3.3.2 and Theorem 3.3.4.

Theorem 3.3.4 [IDL $\rightarrow$ XIDL, Standard Model] Let $\mathbb{G}$ be a group of prime order $p$ with generator $g$. Let $q_1, q_2$ be positive integers. Let $A_{xidl}^{\text{dl}}$ be an adversary against $G_{\mathbb{G}, g, q_1, q_2}^{xidl}$. Then, an
adversary \( \mathcal{A}_{\text{idl}} \) can be constructed so that

\[
\text{Adv}_{G,\text{idl}}^{\mathcal{A}_{\text{idl}}}(\cdot) \leq \sqrt{q_2 \cdot \text{Adv}_{G,\text{idl}}^{\mathcal{A}_{\text{idl}}}(\cdot)} + \frac{q_2}{p}.
\]

Furthermore, the running time of \( \mathcal{A}_{\text{idl}} \) is about twice of that of \( \mathcal{A}_{\text{xdl}} \).

The full proof is given in Section 3.13. We now sketch the intuition. Adversary \( \mathcal{A}_{\text{idl}} \) receives \( X \) from game \( G_{\text{idl}} \), and runs adversary \( \mathcal{A}_{\text{xdl}} \), forwarding it \( X \) as the target point. It answers queries to \( \mathcal{A}_{\text{xdl}} \)'s \( Nw\text{Tar} \) oracle using its own \( G_{\text{idl}} \), \( \text{Ch} \) oracle. Specifically, the \( j \)-th query \( S \) to \( Nw\text{Tar} \) is responded to with \( e_j \leftarrow G_{\text{idl}} \cdot \text{Ch}(S) \), and \( \mathcal{A}_{\text{idl}} \) additionally records the group element \( T_j \leftarrow S \cdot X^{e_j} \). It simulates adversary \( \mathcal{A}_{\text{xdl}} \)'s \( \text{Ch} \) oracle locally, meaning the \( i \)-th query \( \text{Ch}(j_i, R) \) is responded to with a fresh challenge \( c_i \leftarrow \mathbb{Z}_p \). Eventually, adversary \( \mathcal{A}_{\text{idl}} \) gives a response \( I, z \). Our \( \mathcal{A}_{\text{idl}} \) adversary wins game \( G_{\text{idl}} \) if it can produce the discrete log of \( T_j \) for any \( j \) of its choice. To do so, \( \mathcal{A}_{\text{idl}} \) uses rewinding, the analysis of which uses the Forking Lemma [14] that we recall as Lemma 3.9.1. Rewinding is used to produce another response, \((I', z')\), from a forked execution of \( \mathcal{A}_{\text{idl}} \). The Forking Lemma applies to an execution of an algorithm making queries to one oracle, but adversary \( \mathcal{A}_{\text{idl}} \) has two oracles \( Nw\text{Tar} \) and \( \text{Ch} \). We only “fork” \( \mathcal{A}_{\text{idl}} \) on its queries to \( \text{Ch} \). Specifically, we program oracle \( Nw\text{Tar} \) to behave identically compared to the first run (meaning we use previously recorded values of \( e_1, \ldots \) as long as they are defined). In the second run, oracle \( \text{Ch} \) is replied with \( c_1, \ldots, c_{I-1}, c'_I, \ldots \), where \( c'_I, \ldots \) are randomly chosen from \( \mathbb{Z}_p \). Let us assume that \( \mathcal{A}_{\text{idl}} \) has derived two valid responses from \( \mathcal{A}_{\text{idl}} \) using the Forking Lemma. Then it is guaranteed that \( I = I' \) and \( c_I \neq c'_I \). Moreover, we know the two executions of \( \mathcal{A}_{\text{idl}} \) only differ after the response of the \( I \)-th query to \( \text{Ch} \), so the \( I \)-th query to \( \text{Ch} \) in both runs is some \( J, R_I \). This allows our adversary to solve the equations \( g^z = R_I \cdot T^{c_I} \) and \( g'^z = R_I \cdot T^{c'_I} \) (which are guaranteed to be true if both runs succeed) to compute \( DL(G, g)(T_J) \) and thus win the IDL game.
3.4 Definitions for multi-signatures

DEFKLNS [40] found subtle gaps in some prior proofs of security for some two-round multi-signature schemes [8, 62, 85]. Some of the latter schemes had been around for quite a long time before this happened. This suggests that, in the domain of multi-signatures, we need more care and careful analyses. We suggest that this needs to begin with definitions. The ones in prior work, stemming mostly from [14], suffer from some lack of detail and precision. In particular, the very syntax of a multi-signature scheme is not specified in detail. This results in scheme descriptions that lack somewhat in precision, and to proofs that stay at a high level in part due to lack of technical language in which to give details. This in turn can lead to bugs.

To address these issues, we revisit the definitions. We start with a detailed syntax that formalizes the signing protocol as a stateful algorithm, run separately by each player. (The state will be maintained by the overlying game.) Details addressed include that a player knows its position in the signer list, that player identities are separate from public keys, and integration of the ROM through a parameter describing the type of ideal hash functions needed. Then we give a security definition written via a code-based game.

**Syntax.** A multi-signature scheme \( MS \) specifies algorithms \( MS.Kg \), \( MS.Vf \), \( MS.Sign \), as well as a set \( MS.HF \) of functions, and an integer \( MS.nr \), whose intent and operation is as follows. **Key generation.** Via \( (pk, sk) \leftarrow MS.Kg \), the key generation algorithm generates public signature-verification key \( pk \) and secret signing key \( sk \) for a user. (Each user is expected to run this independently to get its keys.) **Hash functions.** \( MS.HF \) is a set of functions, from which, via \( h \leftarrow MS.HF \), one is drawn and provided to scheme algorithms (except key generation) and the adversary as the random oracle. Specifying this as part of the scheme allows the domain and range of the random oracle to be scheme-dependent. **Verification.** Via \( d \leftarrow MS.Vf^H(pk, m, \sigma) \), the verification algorithm deterministically outputs a decision \( d \in \{\text{true}, \text{false}\} \) indicating whether or not \( \sigma \) is a valid signature on message \( m \) under a vector \( pk \) of verification keys. **Signing.**
The signing protocol is specified by signing algorithm MS.Sign. In each round, each party, applies this algorithm to its current state $\text{st}$ and the vector $\text{in}$ of received messages from the other parties, to compute an outgoing message $\sigma$ (viewed as broadcast to the other parties) and an updated state $\text{st}'$, written $(\sigma, \text{st}') \leftarrow \text{MS.Sign}^H(\text{in}, \text{st})$. In the last round, $\sigma$ is the signature that this party outputs. (See Figure 3.4.)

**Rounds.** The interaction consists of a fixed number $\text{MS.nr}$ of rounds. (We number the rounds 0, ..., $\text{MS.nr}$. The final broadcast of the signature is not counted as in practice it is a local output.)

**Key Aggregation.** We say that a multi-signature scheme $\text{MS}$ supports key aggregation if $\text{MS}$ has additional two algorithms $\text{MS.Ag}$ and $\text{MS.VfAg}$ such that: (1) Via $\text{apk} \leftarrow \text{MS.Ag}^H(pk_1, \ldots, pk_n)$, $\text{MS.Ag}$ generates an aggregate public key, (2) Via $d \leftarrow \text{MS.VfAg}^H(\text{apk}, m, \sigma)$, the aggregate verification algorithm deterministically outputs a decision $d \in \{\text{true}, \text{false}\}$, and (3) the verification algorithm $\text{MS.Vf}$ is defined exactly as $\text{MS.Vf}^H(pk, m, \sigma) := \text{MS.VfAg}^H(\text{MS.Ag}^H(pk), m, \sigma)$.

Some conventions will aid further definitions and scheme descriptions. A party’s state $\text{st}$ has several parts: $\text{st.n}$ is the number of parties in the current execution of the protocol; $\text{st.me} \in [1..\text{st.n}]$ is the party’s own identity; $\text{st.rnd} \in [0..\text{MS.nr}]$ is the current round number; $\text{st.sk}$ is the party’s own signing key; $\text{st.pk}$ is the $\text{st.n}$-vector of all verification keys; $\text{st.msg}$ is the message being signed; $\text{st.rej} \in \{\text{true}, \text{false}\}$ is the decision to reject (not produce a signature) or accept. It is assumed and required that each invocation of $\text{MS.Sign}$ leaves all of these unchanged except for $\text{st.rnd}$, which it increments by 1, and $\text{st.rej}$, which is assumed initialized to false and may at some point be set to true. The state can, beyond these, have other components that vary from protocol to protocol. (For example, Figure 3.5 describing the BN scheme has $\text{st.R}[j], \text{st.f}[j], \text{st.z}[j], \text{st.R}, \ldots$)

We write $\text{st} \leftarrow \text{StInit}(j, \text{sk}, pk, m)$ to initialize $\text{st}$ by setting $\text{st.n} \leftarrow |pk|; \text{st.me} \leftarrow j; \text{st.rnd} \leftarrow 0; \text{st.sk} \leftarrow \text{sk}; \text{st.pk} \leftarrow pk; \text{st.msg} \leftarrow m; \text{st.rej} \leftarrow \text{false}$. If an execution $(\sigma, \text{st}') \leftarrow \text{MS.Sign}^H(\text{in}, \text{st})$ returns $\sigma = \bot$ then it is assumed and required that further executions starting from $\text{st}'$ all return $\bot$ as the output message.

**Correctness.** Algorithm $\text{Exec}_{\text{MS}}$, shown in the left column of Fig. 3.4, executes the
signing protocol of MS on input a vector $sk$ of signing keys, a vector $pk$ of matching verification keys with $|sk| = |pk|$, and a message $m$ to be signed, and with access to random oracle $h \in MS.HF$.

The number of parties $n$ at line 1 is the number of coordinates (length) of $pk$. The state $st_j$ of party $j$ at line 3 is initialized using the function StInit defined above. The loop at line 5 executes $MS.nr$ rounds. Here $b$ denotes the $n$-vector of currently-broadcast messages, meaning $b[i]$ was
broadcast by party \( i \) in the prior round, and the entire vector is the input to party \( j \) for the current round. At line 8, \( b \) now holds the next round of broadcasts.

The correctness game \( G_{MS,n}^{ms-cor} \) shown in the right column of Fig. 3.4 has only one procedure, namely FIN. We say that MS satisfies (perfect) correctness if for all positive integers \( n \) we have \( \Pr[G_{MS,n}^{ms-cor}] = 1 \).

**Unforgeability.** Game \( G_{MS}^{ms-uf} \) in Fig. 3.4 captures a notion of unforgeability for multi-signatures that slightly extends [14]. There is one honest player whose keys are picked at line 1, the adversary controlling all the other players. A new instance of the signing protocol is initialized by calling NS with an index \( k \) and a vector \( pk \) of verification keys that the adversary can choose, possibly dishonestly, subject only to \( pk[k] \) being the verification key \( pk \) of the honest player, as enforced by line 2. The first message of the honest player is sent out, and at this point \( st_{u,rnd} = 1 \). Now the adversary can run multiple concurrent instances of the signing protocol with the honest signer. Oracle \( H \) is the random oracle, simply calling \( h \). Eventually the adversary calls FIN with a forgery index \( k \), a vector of verification keys (subjected to \( pk[k] \) being the public key of the honest signer), a message and a claimed signature. It wins if verification succeeds and the forgery was non-trivial. The ms-uf-advantage of adversary \( \mathcal{A} \) is \( \text{Adv}_{MS}^{ms-uf}(\mathcal{A}) = \Pr[G_{MS}^{ms-uf}(\mathcal{A})] \).

It is convenient for (later) proofs to have a separate signing oracle \( \text{SIGN}_j \) for each round \( j \in [1..MS.nr] \). It is required that any \( \text{SIGN}_j(s,\cdot) \) satisfy \( s \in [1..u] \), and that the prior round queries \( \text{SIGN}_k(s,\cdot) \) for \( k < j \) have already been made. It is required that for each \( j,s \), at most one \( \text{SIGN}_j(s,\cdot) \) query is ever made.

**Remarks.** Our syntax and security notions for multi-signatures view a group of signers as captured by the vector (rather than the set) of their public keys. So for example, a forgery \( ((pk_1,pk_2),m,\sigma) \) is considered to be non-trivial even if there was a previous signing session under public keys \( (pk_2,pk_1) \) and message \( m \). This differs from previous formalizations that work instead with sets of public keys. However, previous definition can be recovered if a canonical encoding of sets of public keys into vectors of public keys is fixed in the usage of a scheme.
3.5 Analysis of the BN scheme

BN scheme. Let \( G \) be a group of prime order \( p \). Let \( g \) be a generator of \( G \) and let \( \ell \geq 1 \) be an integer. The associated BN \([14]\) multi-signature scheme \( MS = BN[G, g, \ell] \) is shown in detail, in our syntax, in Fig. 3.5. The set \( MS.HF \) consists of all functions \( h \) such that \( h(0, \cdot) : \{0, 1\}^* \rightarrow \{0, 1\}^\ell \) and \( h(1, \cdot) : \{0, 1\}^* \rightarrow \mathbb{Z}_p \). For \( b \in \{0, 1\} \) we write \( H_b(\cdot) \) for \( H(b, \cdot) \), so that scheme algorithms, and an ms-uf adversary, will have access to oracles \( H_0, H_1 \) rather than just \( H \).

The signing protocol has 3 rounds. In round 0, player \( j \) picks \( r \leftarrow \mathbb{Z}_p \), stores \( g^r \) in its state as \( st.R[j] \), computes, and stores in its state, a value \( st.t[j] \leftarrow H_0((j, st.R[j])) \) that we call the BN-commitment, and broadcasts the BN-commitment. (Per our syntax, what is returned is the message to be broadcast and the updated state to be retained.) Since each player does this, in round 1, player \( j \) receives the BN-commitments of the other players, storing them in vector \( st.t \), and now broadcasting \( st.R[j] \). In round 2, these broadcasts are received, so player \( j \) can form the vector \( st.R \). At line 20, it returns \( \perp \) if one of the received values fails to match its commitment. As per our conventions, when this happens, this player will always broadcast \( \perp \) in the future, so for round 3 we assume lines 21 and 22 are executed. These lines create the second component \( st.z[j] \) of a Schnorr signature relative to the Schnorr-commitment \( st.R[j] \) defined at line 13, and the player’s own secret key, the computations being modulo \( p \). This \( st.z[j] \) is broadcast, so that, in round 3, our player receives the corresponding values from the other players. At line 27 it forms their modulo-\( p \) sum \( z \) and then forms the final signature \((st.R, z)\).

Our description of the signing protocol differs, from that in \([14]\), in some details that are brought out by our syntax, for example in using explicit party identities rather than seeing these as implicit in public keys.

Prior bounds. We recall the prior result of \([14]\). Let \( MS = BN[G, g, \ell] \) and let \( A_{ms} \) be an adversary for game \( G_{MS}^{ms-uf} \). Assume the execution of game \( G_{MS}^{ms-uf} \) with \( A_{ms} \) has at most \( q \) distinct queries across \( H_0, H_1 \) and at most \( q_s \) queries to NS. Suppose the number of parties
Figure 3.5: Algorithms of the multi-signature scheme $\text{BN}[G, g, \ell]$ and $\text{MuSig}[G, g, \ell]$, where $G$ is a group of prime order $p$ with generator $g$. Code that differs between the two schemes is marked explicitly. Oracle $H_i(\cdot)$ is defined to be $H(i, \cdot)$ for $i = 0, 1$ (BN) and $i = 0, 1, 2$ (MuSig). (length of verification-key vector) in queries to NS and FIN is at most $n$. Let $a = 8q_s + 1$ and
\[ b = 2q + 16n^2q_s. \] Let \( p = |G| \). Then BN \[14\] give a DL-adversary \( A_{dl} \) such that

\[
\text{Adv}_{\text{MS}}^{\text{ms-uf}}(A_{ms}) \leq \sqrt{(q + q_s) \cdot \left( \text{Adv}_{G, g}^{\text{dl}}(A_{dl}) + \frac{a}{p} + \frac{b}{2^\ell} \right)}. \tag{3.2}
\]

The running time of \( A_{dl} \) is twice that of the execution of game \( G_{\text{MS}}^{\text{ms-uf}} \) with \( A_{ms} \). BN obtain this result via their general forking lemma, which uses rewinding and accounts for the square-root in the bound.

**SECURITY OF BN FROM IDL.** We give a IDL \(\rightarrow\) BN reduction that is **tight** and in the **standard model**. Combining this with our tight AGM reduction DL \(\rightarrow\) IDL of Theorem 3.3.1 we conclude a tight AGM reduction DL \(\rightarrow\) BN. However, the standard model tight IDL \(\rightarrow\) BN reduction is also interesting in its own right. It says that BN is just as secure as the Schnorr identification scheme. Since the latter has been around and resisted cryptanalysis for quite some time, this is good support for the security of BN.

**Theorem 3.5.1** [IDL \(\rightarrow\) BN, Standard Model] Let \( G \) be a group of prime order \( p \). Let \( g \) be a generator of \( G \) and let \( \ell \geq 1 \) be an integer. Let \( MS = \text{BN}[G, g, \ell] \) be the associated BN multisignature scheme. Let \( A_{ms} \) be an adversary for game \( G_{\text{MS}}^{\text{ms-uf}} \) of Figure 3.4. Assume the execution of game \( G_{\text{MS}}^{\text{ms-uf}} \) with \( A_{ms} \) has at most \( q_0, q_1, q_s \) distinct queries to \( H_0, H_1, \text{NS} \), respectively, and the number of parties (length of verification-key vector) in queries to \( \text{NS} \) and \( \text{FIN} \) is at most \( n \). Let \( \alpha = q_s(4q_0 + 2q_1 + q_s) \) and \( \beta = q_0(q_0 + n) \). Then we construct an adversary \( A_{idl} \) for game \( G_{G, g, q_1}^{\text{idl}} \) (shown explicitly in Figure 3.17) such that

\[
\text{Adv}_{\text{MS}}^{\text{ms-uf}}(A_{ms}) \leq \text{Adv}_{G, g, q_1}^{\text{idl}}(A_{idl}) + \frac{\alpha}{2^\ell} + \frac{\beta}{2^\ell}. \tag{3.3}
\]

The running time of \( A_{idl} \) is about that of the execution of game \( G_{\text{MS}}^{\text{ms-uf}} \) with \( A_{ms} \). Furthermore, adversary \( A_{idl} \) is algebraic if adversary \( A_{ms} \) is.

Above, \( q_0 \) is the number of distinct queries to \( H_0 \) made, not directly by the adversary, but across
the execution of the adversary in game $G^{ms\text{-uf}}_{MS}$, and similarly for $q_1$. A lower bound on $q_1$ is the length of $pk$ in $A_{ms}$’s FIN query, so we can assume it is positive. With the above theorem, we can now derive an upper bound $UB^{ms\text{-uf}}_{MS}(t, q, q_s, p)$ of the advantage of any MS adversary with running time $t$, making $q$ queries to $H$, and $q_s$ signing interactions. We take $\ell \approx \log_2(p)$ and assume that $q_s \leq q \leq t \leq p$. Additionally, we assume that the advantage of any IDL adversary with running time $t$ is at most $t^2/p$ (as justified by Theorem 3.3.2). We obtain $UB^{ms\text{-uf}}_{MS}(t, q, q_s, p) \leq t^2/p$ as shown in Fig. 3.1.

The full proof of Theorem 3.5.1 is given in Section 3.14. Here we give a sketch. The reduction adversary $A_{idl}$ receives a group element $X$ from $G_{C,g,q_1}^{idl}$ and forwards it to adversary $A_{ms}$ as the target public key. In order to run adversary $A_{ms}$, our adversary needs to be able to simulate the signing oracles $NS, Sign_1, Sign_2$ as well as random oracles $H_0$ and $H_1$ without knowing $DL_{C,g}(X)$. We first describe how the reduction proceeds if $A_{ms}$ makes no queries to $NS, Sign_1$ or $Sign_2$, as this steps constitutes the main difference between our proof and the original proof of security for BN [14]. Adversary $A_{idl}$ uses the challenge oracle $G_{C,g,q_1}^{idl}Ch$ to program the random oracle $H_1$ (hence $Ch$ needs to be able to be queried up to the number of times $H_1$ is evaluated). In particular, for each query $H_1((k, R, pk, m))$ where $pk[k] = X$, our adversary first computes $T \leftarrow R \cdot \prod_{j \neq k} pk[j]^{H_1((j, R, pk, m))}$, then obtains $c \leftarrow Ch(T)$ before returning $c$ as the return value for the query $H_1((k, R, pk, m))$. By construction, a valid forgery for $pk, m$ is some signature $\sigma = (R, z)$ such that

$$g^z = R \cdot \prod_{i=1}^{n} pk[i]^{H_1((i, R, pk, m))} = T \cdot X^c,$$

where the first equality is by the verification equation of BN and the second equality is by the way $H_1$ is programmed. This means that adversary $A_{idl}$ can simply forward the value of $z$ from a valid forgery, along with the index of the $Ch$ query corresponding to the $H_1$ query of the forgery, to break game $G_{C,g,q_1}^{idl}$. Moreover, adversary $A_{idl}$ succeeds as long as the forgery given by $A_{ms}$
is valid.

It remains to show that oracles NS, SIGN₁, SIGN₂ can be simulated without knowledge of the secret key, DL_{G,g}(X). Roughly, this is done using the zero-knowledge property of the underlying Schnorr identification scheme as well as by programming the random oracles H₀ and H₁. The original proof by [14] constructs an adversary and argues that it simulates these oracles faithfully if certain bad events do not happen. We take a more careful approach and do this formally via a sequence of seven games and use the code-base game playing framework of [19]. This game sequence incurs the additive loss as indicated in Equation (3.3).

\textbf{Converse.} IDL is not merely some group problem that can be used to justify security of BN tightly; the hardness of IDL is, in fact, tightly equivalent to the MS-UF security of BN. Formally, we give below a reduction turning any adversary against IDL into a forger A_{ms} against BN. This means that any security justification for BN must also justify the hardness of IDL.

\textbf{Theorem 3.5.2 [BN → IDL, Standard Model]} Let G be a group of prime order p. Let g be a generator of G and let ℓ ≥ 1 be an integer. Let MS = BN[G,g,ℓ] be the associated BN multi-signature scheme. Let q be a positive integer and A_{idl} be an adversary against G_{ms}^{idl}. Then, we can construct an adversary A_{ms} for game G_{ms-uf}^{MS}, making no queries to NS, and at most 2q queries to H₁, such that

\[
\text{Adv}_{MS}^{ms-uf}(A_{ms}) \geq \text{Adv}_{G,g,q}^{idl}(A_{idl}).
\] (3.4)

The running time of A_{ms} is about that of A_{idl}.

\textbf{Proof of Theorem 3.5.2:} Consider the adversary given in Fig. 3.6. The adversary receives the target public key pk from the MS-UF game and samples a key pair (pk', sk') ← MS.K_g. The adversary will attempt to forge a signature against the vector of public keys (pk, pk'). Adversary A_{ms} forwards X = pk as the target point and runs IDL adversary A_{idl}. For each query CH(R) of
\[ A_{\text{idl}} \times \rightarrow pk; (pk', sk') \leftarrow \text{MS.Kg}() \]
\[ (I,z) \leftarrow A_{\text{idl}}(pk) \parallel g^z = R_I \cdot pk^{c_{I,1}} \]
\[ \sigma \leftarrow (R_I, z + sk' \cdot c_{I,2} \mod p) ; \text{Return} \ ((pk, pk'), m_I, \sigma) \]

**Figure 3.6:** Adversary \( A_{\text{ms}} \) for Theorem 3.6.1. For an integer \( i, \langle i \rangle \) denote the binary representation of \( i \).

\( A_{\text{idl}}, \) adversary \( A_{\text{ms}} \) simulates the response as per line 4 to 6. If \( A_{\text{idl}} \) succeeds, it must be that

\[ g^z = R_I \cdot pk^{c_{I,1}}. \]

The value of \( z \) can be used to construct a forgery signature (line 3).

### 3.6 Analysis of the MuSig scheme

The current three-round version of MuSig has been proposed and analyzed by both [64] and [25]. Roughly, it is the BN scheme with added key aggregation.

Let \( G \) be a group of prime order \( p \). And let \( g \) be a generator of \( g \) and \( \ell \geq 1 \) be an integer. The formal specification of \( \text{MS} = \text{MuSig}[G, g, \ell] \) in our syntax is shown in Fig. 3.5. There are minimal differences between MuSig and BN and we only highlight the differences. The set \( \text{MS.HF} \) consists of all functions \( h \) such that \( h(0, \cdot) : \{0,1\}^* \rightarrow \{0,1\}^\ell \) and \( h(i, \cdot) : \{0,1\}^* \rightarrow \mathbb{Z}_p \) for \( i = 1,2 \). Verification is done as follows. First, an aggregate key \( apk \) for the list of keys \( pk = (pk_1, \ldots, pk_n) \) is computed as \( apk \leftarrow pk_1^{H_1(1, pk)} \cdots pk_n^{H_2(m, pk)} \) (line 8). Next, a single challenge is derived from the commitment \( R \) and aggregate key \( apk \) (line 9). The signature \( (R, z) \) is valid if \( g^z = R \cdot apk^c \). The second round of signing also changes accordingly to generate a valid
signature (line 24 and 25).

The following gives a tight, standard-model reduction XIDL → MuSig. Combining this with our tight AGM chain DL → IDL → XIDL from Theorems 3.3.1 and 3.3.3, we get a tight AGM reduction DL → MuSig.

**Theorem 3.6.1** [XIDL → MuSig, Standard Model] Let \( G \) be a group of prime order \( p \). Let \( g \) be a generator of \( G \) and \( \ell \geq 1 \) be an integer. Let \( MS = MuSig[G, g, \ell] \) be the associated MuSig multi-signature scheme. Let \( A_{ms} \) be an adversary for game \( G^{ms-uf}_{MS} \) of Figure 3.4. Assume the execution of game \( G^{ms-uf}_{MS} \) with \( A_{ms} \) has at most \( q_0, q_1, q_2, q_s \) distinct queries to \( H_0, H_1, H_2, NS \), respectively, and the number of parties (length of verification-key vector) in queries to NS and \( FIN \) is at most \( n \). Let \( \alpha = q_s(4q_0 + 2q_1 + q_2) + 2q_1q_2 \) and \( \beta = q_0(q_0 + n) \). Then we construct an adversary \( A_{xidl} \) for game \( G^{xidl}_{G, g, q_2, q_1} \) (shown explicitly in Figure 3.23) such that

\[
Adv^{ms-uf}_{MS}(A_{ms}) \leq Adv^{xidl}_{G, g, q_2, q_1}(A_{xidl}) + \frac{\alpha}{2p} + \frac{\beta}{2t}. \tag{3.5}
\]

The running time of \( A_{xidl} \) is about that of the execution of game \( G^{ms-uf}_{MS} \) with \( A_{ms} \). Furthermore, adversary \( A_{xidl} \) is algebraic if adversary \( A_{ms} \) is.

We remark that the values of \( q_1 \) and \( q_2 \) above arise from the number of queries to \( H_1 \) and \( H_2 \) made in the execution of \( G^{ms-uf}_{MS}(A_{ms}) \). As a result, the appearance of \( q_1 \) and \( q_2 \) has their orders “switched” compared to in Section 3.3. With the above theorem, we can now derive an upperbound \( UB^{ms-uf}_{MS}(t, q, q_s, p) \) of the advantage of any MS adversary with running time \( t \), making \( q \) queries to \( H \), and \( q_s \) signing interactions. We take \( \ell \approx \log_2(p) \) and assume that \( q_s \leq q \leq t \leq p \). Additionally, we assume that the advantage of any XIDL adversary with running time \( t \) is at most \( t^2/p \) (as justified by Theorem 3.3.4). We obtain \( UB^{ms-uf}_{MS}(t, q, q_s, p) \leq t^2/p \) as shown in Fig. 3.1.

We again describe the reduction at a high level and defer the full proof to Section 3.15. First, the reduction adversary \( A_{xidl} \) receives group element \( X \) from game \( G^{xidl}_{G, g, q_2, q_1} \) and runs \( A_{ms} \) with the target public key set to \( X \). Similar to the proof of Theorem 3.3.1, our adversary
needs to simulate the signing oracles NS, \text{Sign}_1, \text{Sign}_2 as well as \text{H}_0, \text{H}_1, \text{H}_2 without knowing DL_{G,g}(X) in order to run \mathcal{A}_{ms}. This again relies on the zero-knowledge property of the underlying Schnorr identification scheme and the programming of \text{H}_0, \text{H}_1, \text{H}_2. This step is done formally in a game sequence in the full proof and incurs the additive loss in Equation (3.5). To turn a forgery into a break against XIDL, our adversary programs \text{H}_1 and \text{H}_2 as follows. For the \( j \)-th query of \( \text{H}_2((k, pk)) \) where \( pk[k] = X \), the adversary first computes \( S \leftarrow \prod_{i \neq k} pk[i]^{H_2((i, pk))} \), then obtains \( e_j \leftarrow \text{NwTar}(S) \) before returning \( e_j \) as the response for the query. We remark that this particular query of \( \text{H}_2 \) have created an aggregate public key \( apk = \prod_{i=1}^{pk} pk[i]^{H_2((i, pk))} = S \cdot X^{e_j} \), which is also the value of \( T_j \) that is recorded in the game \( G_{\text{id}_1 G_{\text{id}}, g, q_2, q_1} \). For each \( i \)-th query of \( \text{H}_1((R, apk, m)) \), the adversary first finds the index \( j_{\text{sel}} \) of the \( \text{H}_2 \)-query that corresponds to the input \( apk \), then obtains \( c_i \leftarrow \text{Ch}(j_{\text{sel}}, R) \) before returning \( c_i \) as the response for the query. If the eventual forgery is given for these two particular queries to \( \text{H}_1 \) and \( \text{H}_2 \), meaning forgery is \( pk, m, (R, z) \) for some \( z \), then the verification equation of the signature scheme says that \( g^z = R \cdot apk^{H_1((R, apk, m))} \). But this matches exactly the winning condition of \( G_{\text{id}_1 G_{\text{id}}, g, q_2, q_1} \), since \( apk = T_{j_{\text{sel}}} \) and \( c_i = H_1((R, apk, m)) \). Hence, our adversary \( \mathcal{A}_{\text{id}_1} \) can simply return \( (i, z) \) to break XIDL, as long as the forgery provided by \( \mathcal{A}_{ms} \) is valid.

Similar to the relation between IDL and BN, XIDL is also tightly equivalent to the MS-UF security of MuSig. In particular, we turn any adversary breaking XIDL into a forger against MuSig. This means that any security justification for MuSig must also justify the hardness of XIDL.

**Theorem 3.6.2** [MuSig \( \rightarrow \) XIDL, Standard Model] Let \( G \) be a group of prime order \( p \). Let \( g \) be a generator of \( G \) and let \( \ell \geq 1 \) be an integer. Let \( \text{MS} = \text{MuSig}[G, g, \ell] \) be the associated MuSig multi-signature scheme. Let \( q_1, q_2 \) be a positive integers and \( \mathcal{A}_{\text{id}_1} \) be an adversary against \( G_{\text{id}_1 G_{\text{id}}, g, q_2, q_1} \). Then, we can construct an adversary \( \mathcal{A}_{ms} \) for game \( G_{\text{ms-uf}}^{\text{ms-uf}} \), making no queries to
\[ A^{H_1,H_2}(pk): \]

1. \( X \leftarrow pk ; (I,z) \leftarrow A_{\text{xidl}}^{\text{Ch}}(pk) ; J \leftarrow \text{TI}[I] \)
2. \( \sigma \leftarrow (R_I,z) ; \text{Return } ((pk,S_I),m_I,\sigma) \)

\[ \text{NW} \text{TAR}(S): \]

1. \( j \leftarrow j+1 ; S_j \leftarrow S \)
2. \( e_{j,1} \leftarrow H_2((1,(pk,S))) \); \( e_{j,2} \leftarrow H_2((2,(pk,S))) \); \( e_j \leftarrow e_{j,2}/e_{j,1} \mod p \)
3. \( apk_j \leftarrow pk^{e_{j,1}}S^{e_{j,2}} ; T_j \leftarrow pk \cdot S_j ; \text{Return } e_j \)

\[ \text{Ch}(j_{\text{sel}},R): \]

1. \( i \leftarrow i+1 ; R_i \leftarrow R ; m_i \leftarrow \langle i \rangle ; \text{TI}[i] \leftarrow j_{\text{sel}} \)
2. \( c_i \leftarrow H_1((apk_{j_{\text{sel}}},R,m_i)) \cdot e_{j_{\text{sel}},1} ; \text{Return } c_i \)

**Figure 3.7:** Adversary \( A_{\text{ms}} \) for Theorem 3.6.1. For an integer \( i, \langle i \rangle \) denote the binary representation of \( i \).

NS, and at most 2\( q_1 \) and 2\( q_2 \) queries to \( H_1 \) and \( H_2 \) respectively, such that

\[
\text{Adv}^{\text{ms-uf}}_{\text{MS}}(A_{\text{ms}}) \geq \text{Adv}^{\text{xidl}}_{G,g,q_2,q_1}(A_{\text{xidl}}). \tag{3.6}
\]

The running time of \( A_{\text{ms}} \) is about that of \( A_{\text{xidl}} \).

**Proof of Theorem 3.6.2:** Consider the adversary given in Fig. 3.7. The adversary receives the target public key \( pk \) from the MS-UF game. Adversary \( A_{\text{ms}} \) forwards \( X = pk \) as the target point and runs XIDL adversary \( A_{\text{xidl}} \). For each query \( \text{NW} \text{TAR}(S) \) of \( A_{\text{xidl}} \), adversary \( A_{\text{ms}} \) uses \( S \) as a public key to generate the aggregate key \( apk \) for the list \((pk,S)\). By construction, the \( j \)-th target \( T_j \) for the XIDL game is related to \( apk_j \) by \( apk_j = T_j^{e_{j,1}} \). For each \( \text{Ch}(j_{\text{sel}},R) \) query of \( A_{\text{xidl}} \), adversary \( A_{\text{ms}} \) programs in the \( H_1 \) outputs corresponding to a forgery against the aggregate key \( apk_{j_{\text{sel}}} \) (line 6 and 7). By construction, if \( A_{\text{xidl}} \) succeeds, it must be that

\[
E = R_I \cdot T_j^{e_{j,1}} = R_I \cdot T_j^{H_1((apk_{j_{\text{sel}}},R,m_i)) \cdot e_{j,1}} = R_I \cdot apk_{j_{\text{sel}}}^{H_1((apk_{j_{\text{sel}}},R,m_i))}. \]

Hence, adversary \( A_{\text{ms}} \) produces a valid forgery at line 2.  \( \Box \)
The formal definition of our scheme is given in Theorem 3.7.2. Referring to Fig. 3.2, these results establish arrow 5. We refer to Fig. 3.1 for Fig. 3.8.

HBMS: Our new two-round multi-signature scheme

Recall that BN and MuSig are three-round schemes, and two-round schemes are desired due to blockchain applications. In this section, we introduce our new, efficient two-round multi-signature scheme supporting key-aggregation, HBMS. We first demonstrate its tight security against algebraic adversaries (Theorem 3.7.1), before justifying its security in the standard model (Theorem 3.7.2). Referring to Fig. 3.2, these results establish arrow 5. We refer to Fig. 3.1 for comparisons of HBMS against other two-round schemes.

**Figure 3.8:** Two-round multi-signature scheme \( MS = \text{HBMS}[G,g] \) parameterized by a group \( G \) of prime order \( p \) with generator \( g \).

### 3.7 HBMS: Our new two-round multi-signature scheme


due to blockchain applications. In this section, we introduce our new, efficient two-round multi-signature scheme supporting key-aggregation, HBMS. We first demonstrate its tight security against algebraic adversaries (Theorem 3.7.1), before justifying its security in the standard model (Theorem 3.7.2). Referring to Fig. 3.2, these results establish arrow 5. We refer to Fig. 3.1 for comparisons of HBMS against other two-round schemes.

**TWO-ROUND MS SCHEME HBMS.** The formal definition of our scheme is given in Fig. 3.8. HBMS has the same key generation algorithm \( \text{Kg} \) and key aggregation \( \text{Ag} \) algorithm as
MuSig. We describe informally the process involved to sign a message $m$ under a vector of public keys $pk$. In the first round, each signer $i$ samples $s_i$ and $r_i$ uniformly from $\mathbb{Z}_p$ and computes a commitment

$$T_i \leftarrow H_0((pk, m))^{s_i} \cdot g^{r_i},$$

which is sent to every other signer. In the second round, each signer receives the list of commitments $T_1, \ldots, T_n$ from each signer, and computes the aggregate value $T \leftarrow \prod_i T_i$. Each signer then computes the challenge value as $c \leftarrow H_1((T, apk, m))$. To compute the reply, each signer $i$ computes $z_i \leftarrow r_i + sk \cdot c \cdot H_2((i, pk))$ and sends $(s_i, z_i)$ to every other signer. Finally, any signer can now compute the final signature as $(T, s, z)$ where $s = \sum_i s_i$ and $z = \sum_i z_i$. To verify a signature $(T, s, z)$ on $(pk, m)$, the equation

$$g^z \cdot H_0((pk, m))^s = T \cdot apk^{H_1((T, apk, m))},$$

must hold, where $apk = \prod_{i=1}^{\left|pk\right|} pk[i]^{H_2((i, pk))}$. Compared to MuSig, the verification equation of HBMS involves an additional power of $H((pk, m))$ (hence the name HBMS, or “Hash-Base Multi-Signature”).

**Tight Security Against Algebraic Adversaries.** We first show that HBMS is tightly MS-UF-secure against algebraic adversaries.

**Theorem 3.7.1** [DL → HBMS, AGM] Let $\mathbb{G}$ be a group of prime order $p$ with generator $g$. Let MS be the HBMS[$\mathbb{G}, g$] scheme. Let $A^{alg}_{ms}$ be an algebraic adversary for game $G^{ms-uf}_{MS}$ of Figure 3.4 Assume the execution of game $G^{ms-uf}_{MS}$ with $A_{ms}$ has at most $q_1, q_2$ distinct queries to $H_1, H_2$, respectively. Then we construct an adversary $A_{dl}$ for game DL[$\mathbb{G}, g$] (shown explicitly in Figure 3.25) such that

$$\text{Adv}^{ms-uf}_{MS}(A^{alg}_{ms}) \leq \text{Adv}^{dl}_{\mathbb{G}, g}(A_{dl}) + \frac{(q_1 + 1)q_2}{p}. \quad (3.7)$$
The running time of $\mathcal{A}_{dl}$ is about that of the execution of game $G_{MS}^{ms-uf}$ with $\mathcal{A}_{ms}^{alg}$.

Above, a reduction is given directly from DL, and there is no multiplicative loss. As before, assuming $q_s \leq q \leq t \leq p$ and the generic hardness of DL (advantage of $t$-time adversary to be at most $t^2/p$), we derive that $UB_{MS}^{ms-uf}(t, q, q_s, p) \leq t^2/p$, as shown in Fig. 3.1.

We give the high level proof sketch here and defer the full proof to Section 3.16. Let $\mathcal{A}_{ms}$ be the algebraic adversary against HBMS. Our reduction adversary $\mathcal{A}_{dl}$ sets its own target point $X$ (which it needs to obtain the discrete log of) as the target public key for $\mathcal{A}_{ms}$. In order to run $\mathcal{A}_{ms}$, our adversary $\mathcal{A}_{dl}$ needs to be able to simulate oracles $\text{NS}, \text{SIGN}_1, \text{SIGN}_2$ (oracles representing the honest signer) as well as random oracles $H_0, H_1, H_2$. We first tackle the problem of simulating the honest signer without knowledge of the corresponding secret key. This is done by programming of random oracle $H_0$. Suppose for $pk, m$, we set $H_0((pk, m))$ to be $h = g^\alpha pk^\beta$ for some $\alpha, \beta \neq 0 \in \mathbb{Z}_p$ (whose exact distribution will be specified later). When the adversary interacts with the honest signer, the honest signer must first provide some commitment $T \in G$ (in the output of $\text{NS}$), then later produce $z, s \in \mathbb{Z}_p$ (in the output of $\text{SIGN}_1$) such that

$$g^zh^s = T \cdot pk^c,$$  \hspace{1cm} (3.8)

where $c \in \mathbb{Z}_p$ is some challenge value (that is derived using the random oracle and the responses of the adversary). To do this, our adversary sets commitment $T = g^ah^b$ for $a, b \leftarrow \mathbb{Z}_p$. It shall be convenient to express $pk$ in terms of $g$ and $h$ as well. Note that as long as $\beta \neq 0$, $pk = h^{(\beta^{-1})}g^{-\alpha(\beta^{-1})}$. Since both $T$ and $pk$ are known to be of the form $g^*h^s$ (where $*$ denotes some element of $\mathbb{Z}_p$), so is the group element $T \cdot pk^c$ (for any known value of $c$). Hence, the right-hand side of Equation (3.8) is of the form $g^zh^s$ for some values $z$ and $s$ that our adversary can compute, and our adversary can return them as response in the second round. Above, we noted that this works as long as $\beta \neq 0$. To guarantee this, we sample $\alpha \leftarrow \mathbb{Z}_p$ and $\beta \leftarrow \mathbb{Z}_p^*$ in $H_0$. It remains to check that such way of simulating the honest signer is indistinguishable from the behavior of
an honest signer holding the secret key and executing the protocol. Roughly, this is because in both cases, the triple \((T, z, s)\) is uniformly distributed over \(\mathbb{G} \times \mathbb{Z}_p^2\), subjected to the condition that Equation (3.8) holds.

Now, our adversary \(A_{dl}\) can move onto turning a forgery from \(A_{ms}\) into a discrete logarithm for target point \(X\). Suppose adversary \(A_{ms}\) returns forgery \((pk, m, (T, s, z))\). Then,

\[ g^zh^s = T \cdot apk^c, \quad (3.9) \]

where \(apk = \prod_{i=1}^{|pk|} pk[i]^{H_2((i, pk))}\) and \(c = H_1((T, apk, m))\). Since \(A_{ms}\) is algebraic, our adversary \(A_{dl}\) can rewrite Equation (3.9) to the form \(g^{\alpha_s} = X^{\alpha_x}\), which allows us to compute the discrete log of \(X\) as \(\alpha_g \alpha_s^{-1} \mod p\), as long as \(\alpha_x\) is not zero. The full proof upperbounds the probability that \(\alpha_x = 0\) to be at most \(q_1q_2/p\). Outside of this bad event, our adversary \(A_{dl}\) will successfully compute the value of \(\text{DL}_{G,g}(X)\) from a valid forgery.

**Standard Model Security of HBMS.** We reduce the security of HBMS to the hardness of XIDL, with factor \(q_s\) loss. For applications, the number of signing queries \(q_s\) is much less than adversarial hash function evaluations. As a result, even though our reduction here is non-tight, the reduction loss is smaller compared to previous results for BN, MuSig or other two round schemes (cf. Figure 3.1 and 3.1), at the expense of assuming the hardness of XIDL. Interestingly, due to Theorem 3.6.2 our results also state that HBMS is secure as long as MuSig is (via the reduction chain MuSig \(\rightarrow\) XIDL \(\rightarrow\) HBMS), and this reduction again only losses a factor of \(q_s\) in the advantage.

**Theorem 3.7.2 [XIDL \(\rightarrow\) HBMS, Standard Model]** Let \(\mathbb{G}\) be a group of prime order \(p\) with generator \(g\). Let MS be the HBMS[\(\mathbb{G}, g\)] scheme given in Fig. 3.8. Let \(A_{ms}\) be an adversary for game G_{MS}^{ms-uf} of Figure 3.4. Assume the execution of game G_{MS}^{ms-uf} with \(A_{ms}\) has at most \(q_0, q_1, q_2, q_s\) distinct queries to \(H_0, H_1, H_2, \text{NS}\), respectively. Then we construct an adversary
\[ \mathcal{A}_{\text{Xidl}} \text{ for game } G_{\text{ms}}^{\text{Xidl}, q_2, q_1} \text{ (shown explicitly in Figure 3.27)} \text{ such that} \]

\[
\text{Adv}_{\text{ms}}^{\text{ms}-\text{uf}}(\mathcal{A}_{\text{ms}}) \leq e(q_s + 1) \cdot \text{Adv}_{\mathcal{G}_{\text{ms}}^{\text{Xidl}, q_2, q_1}}(\mathcal{A}_{\text{Xidl}}) + \frac{q_1 q_2}{p}, \tag{3.10}
\]

where \( e \) is the base of the natural logarithm. Adversary \( \mathcal{A}_{\text{Xidl}} \) makes \( q_2 \) queries to \( \text{NwTAR} \) and \( q_1 \) queries to \( \text{Ch} \). The running time of \( \mathcal{A}_{\text{Xidl}} \) is about that of the execution of game \( G_{\text{ms}}^{\text{ms}-\text{uf}} \) with \( \mathcal{A}_{\text{ms}} \).

Concretely, if we assume that XIDL is quantitatively as hard as DL, then against any adversary with running time \( t \), making \( q \) evaluations of the random oracle and making at most \( q_s \) signing queries, HBMS has security \( (q_t^2 + q_s^2)/p \approx q_s t^2/p \).

We sketch the highlevel proof here and give the full proof in Section 3.17. Our adversary receives the target point \( X \) from the XIDL game and sets it as the target public key for adversary \( \mathcal{A}_{\text{ms}} \). As before, in order to run \( \mathcal{A}_{\text{ms}} \), we need to simulate oracles \( \text{NwTAR}, \text{SIGN}_1, \text{SIGN}_2 \) as well as \( H_0, H_1, H_2 \). Recall that in the AGM proof, we can simulate the honest signer for \( \text{SIGN}_p \) if we set \( H_0((\text{pk}, m)) = g^\alpha \). However, this way of programming \( H_0 \) does not facilitate in turning a forgery into a break for XIDL. Instead, we would like to program \( H_0((\text{pk}, m)) = g^\alpha \) for the forgery \( \text{pk}, m \). To do this, we use a technique of Coron [32], which programs \( H_0((\text{pk}, m)) \) randomly in one of these two ways depending on a biased coin flip (with probability \( \rho \) of giving 1). The reduction only succeeds if correct “guesses” are made. Specifically, we need that for every \( \text{pk}, m \) that is queried to the honest signer (in NS) then \( H_0((\text{pk}, m)) \) must have been programmed to be \( g^\alpha \text{pk}^\beta \) (for some \( \alpha \) and \( \beta \)), and for the forgery \( \text{pk}, m \), it must be that \( H_0((\text{pk}, m)) = g^\alpha \) (for some \( \alpha \)). We can then optimize for the value of \( \rho \), resulting in a multiplicative loss of \( e(1 + q_s) \).

Suppose adversary \( \mathcal{A}_{\text{ms}} \) returns a forgery \( (\text{pk}, m, (T, s, z)) \) where we have previously programmed \( H_0((\text{pk}, m)) = g^\alpha \). The verification equation say that \( g^zh^s = T \cdot \text{ap}k^c \). Since \( h \) is just a power of \( g \), the left-hand side of the verification equation is also a known power of \( g \) (specifically \( g^{z+\alpha \cdot s} \)). This means that our adversary \( \mathcal{A}_{\text{Xidl}} \) can proceed exactly as the reduction for MuSig. In particular, for the \( j \)-th query of \( H_2((k, \text{pk})) \) where \( \text{pk}[k] = X \), the adversary first computes
$S \leftarrow \prod_{i \neq k} pk[i]^{|H_2((i, pk))|}$, then obtains $e_j \leftarrow \text{NW TAR}(S)$ before returning $e_j$ as the response for the query. We remark that this particular query of $H_2$ have created an aggregate public key $apk = \prod_{i=1}^{p_k} pk[i]^{|H_2((i, pk))|} = S \cdot X^{e_j}$, which is also the value of $T_j$ that is recorded in the game $Gm_{\text{XIDL}}^{\text{Gm}, g, H_2, q_1}$. For each $i$-th query of $H_1((T, apk, m))$, the adversary first finds the index $j_{\text{sel}}$ of the $H_2$-query that corresponds to the input $apk$, then obtains $c_i \leftarrow \text{CH}(j_{\text{sel}}, T)$ before returning $c_i$ as the response for the query. If the eventual forgery is given for these two particular queries to $H_1$ and $H_2$, meaning forgery is $pk, m, (T, s, z)$, then the verification equation of the signature scheme says that $g^{z + \alpha \cdot s} = T \cdot apk^{H_1((T, apk, m))}$ (if we programmed $H_0((pk, m))$ to be $g^\alpha$). Hence, our adversary $A_{\text{XIDL}}$ can simply return $(i, z + \alpha \cdot s)$ to break XIDL, as long as the forgery provided by $A_{\text{ms}}$ is valid and we have made the right guesses in programming $H_0$.

### 3.8 Security bounds of multi-signature schemes

We survey previous results on discrete-log-based multi-signature schemes, with a focus on their reduction loss. We restate these results in the same notation and framework to facilitate comparisons. We have used this to obtain the estimates in Figures 3.1 and 3.1.

For the rest of the section, fix a group $G$ of prime order $p$ that shall be used by each of the schemes of interest. Additionally, we assume that we fix adversaries $A_{\text{ms}}$ attacking each multi-signature scheme of interest, with running time $t$ (this is the total execution time of $G_{\text{MS-UF}}(A_{\text{ms}})$ and includes the running time of all oracles), making $q$ queries to the random oracle, $q_s$ queries to NS involving maximum of $N$-signers while achieving success advantage of $\varepsilon$. For convenience, we let $q_T = 1 + q + q_s$.

Bellare and Neven [14] gave a 3-round MS scheme that is based on the DL problem. In particular, they showed that given an MS-UF adversary $A$, there exists DL-adversary with
running time $t'$ achieving success advantage $\varepsilon'$:

$$
\varepsilon' \geq \frac{\varepsilon^2}{q + q_s} - \frac{2q + 16N^2q_s}{2^\ell} - \frac{8Nq_s}{p},
$$

(3.11)

$$
t' \approx 2t,
$$

(3.12)

where $\ell$ is a parameter, describing the output lengths of the random oracle used for commitments.

**MuSig.** BDN [25] and MPSW [64] gave a 3-round MS scheme that adds key aggregation on-top of BN. For security, BDN showed [25][Theorem 4] that given an MS-UF adversary $A$, there exists DL-adversary with running time $t'$ achieving success advantage $\varepsilon'$ where

$$
\varepsilon' = \frac{\varepsilon - \delta}{64},
$$

(3.13)

$$
t' = 512 \cdot t \cdot q^2 T (\varepsilon - \delta)^{-1} \ln^{-2} (64/(\varepsilon - \delta)) ,
$$

(3.14)

$$
\delta = \frac{4Nq_T}{p},
$$

(3.15)

as long as $p > 8q/\varepsilon$. MPSW gave a tighter result by two direct applications of the forking lemma. In particular, they showed that [64][Theorem 1] given an MS-UF adversary $A_{ms}$, there exists DL-adversary with running time $t'$ achieving success advantage $\varepsilon'$ where

$$
\varepsilon' = \frac{\varepsilon^4}{q^3 T} - \frac{16q_s(q + N \cdot q_s)}{p} - \frac{16(q + Nq_s)^2}{2^\ell} + 3,
$$

(3.16)

$$
t' \approx 4t.
$$

(3.17)

**mBCJ.** DEFKLNS [40] gave a 2-round MS scheme mBCJ. For security, they showed that given an MS-UF adversary $A$, there exists DL-adversary with running time $t'$ achieving success
advantage \( \varepsilon' \) where

\[
\varepsilon' = \frac{\varepsilon}{8e(q_s+1)},
\]

\[
t' = t \cdot 64(N+1)^2 q_T(q_s+1) \varepsilon^{-1} \ln^{-1}(8e(N+1)(q_s+1)/\varepsilon),
\]

as long as \( p > 64e(N+1)q_T(q_s+1)/\varepsilon \).

**MuSig-DN.** NRSW [71] gave a 2-round MS scheme that has deterministic signing. For security, their result [71][Theorem 1] roughly translates to: given an an adversary attack MuSig-DN, there exists OMDL adversary attacking DL with success advantage approximately

\[
\varepsilon' \geq \left( \varepsilon - q_s \delta - \frac{q_T^2}{2^{\lambda-2}} - \frac{2}{2^{\lambda/4}} \right)^4 q_T^{-3},
\]

\[
t' \approx 4t,
\]

where \( \lambda \) is a parameter of the scheme and \( \delta \) is a small constant associated with the group.

**MuSig2.** NRS [68] gave a 2-round MS scheme, parameterized by \( \nu \). For \( \nu \geq 4 \), they showed that if there exists \( \mathcal{A} \) attacking their scheme, they [68][Theorem 1] can build \( \nu q_s \)-OMDL adversary with running time \( t' \) achieving success advantage \( \varepsilon' \) where

\[
\varepsilon' \geq \frac{\varepsilon^4}{m^3} - \frac{11}{p} - \frac{43m^4}{(p-1)^{\nu-3}},
\]

\[
t' \approx 4t,
\]

\[
m = (\nu-1)(q+q_s) + 1.
\]

For \( \nu = 2 \), they give a tighter proof against algebraic adversaries. In particular, given an algebraic adversary \( \mathcal{A} \) attacking their scheme for \( \nu = 2 \), they build adversary \( \mathcal{B} \) against \( q_s \)-OMDL that runs
in time $t'$ to achieve success advantage $\epsilon'$ with

$$\epsilon' \geq \epsilon - 14\frac{q^3}{p},$$

$$t' \approx t + O(q^3).$$

**DWMS.** Alper and Burdges [3] gave a 2-round MS scheme DWMS similar to MuSig2 that is also proved secure from OMDL in AGM. Their proof as given is non-concrete. However, tracing through their reduction, we obtained the following reduction loss: given an algebraic MS-UF adversary $\mathcal{A}_{ms}^{alg}$, an $q_s$-OMDL adversary can be constructed with advantage $\epsilon'$ with running time $t'$ where

$$\epsilon' \geq \epsilon - \frac{q^3q^2}{\sqrt{p}},$$

and $t' \approx t$.

### 3.9 Forking lemma

We recall the general forking lemma of [14]. We restate it using the games of Figure 3.9. Each game has just one procedure, Fin, which takes no inputs. The games are parameterized by an algorithm $\mathcal{A}$ that is executed inside the game, and also by an algorithm $\mathcal{I}G$ called an input generator.

**Lemma 3.9.1** [14] Let $q \geq 1$ be an integer. Let $C$ be a set of size $|C| \geq 2$. Let $\mathcal{A}$ be a randomized algorithm that on inputs $x, c_1, \ldots, c_q$ returns a pair, the first element of which is an integer in the range $0, \ldots, q$, and the second element of which we refer to as a side output. Let $\mathcal{I}G$ be a randomized algorithm that, as above, we call the input generator. Consider $G_{m_0}$ (called the
\begin{itemize}
    \item \textbf{Game Gm}_0
      \begin{enumerate}
        \item $x \leftarrow \text{IG}$
        \item $c_1, \ldots, c_q \leftarrow C$
        \item $(I, \sigma) \leftarrow \mathcal{A}(x, c_1, \ldots, c_q)$
        \item Return $(I > 0)$
      \end{enumerate}

    \item \textbf{Game Gm}_1
      \begin{enumerate}
        \item $x \leftarrow \text{IG}$
        \item $\rho \leftarrow \text{rand}(\mathcal{A})$; $c_1, \ldots, c_q \leftarrow C$
        \item $(I, \sigma) \leftarrow \mathcal{A}(x, c_1, \ldots, c_q)$
        \item If $(I = 0)$ then return $(0, \varepsilon, \varepsilon)$
        \item $c'_1, \ldots, c'_q \leftarrow C$
        \item $(I', \sigma') \leftarrow \mathcal{A}(x, c_1, \ldots, c_{I-1}, c'_I, \ldots, c'_q)$
        \item Return $((I = I')$ and $(c_I \neq c'_I))$
      \end{enumerate}
\end{itemize}

\textbf{Figure 3.9:} Games referred to in Lemma 3.9.1. Both games have just one procedure, \texttt{FIN}, which does not take any input. These games run an algorithm \texttt{A} internally.

\textit{single run) and Gm}_1 (called the forked run) given in Fig. 3.9 Then:

\[
\Pr[Gm_0] \leq \frac{q}{|C|} + \sqrt{q \cdot \Pr[Gm_1]} .
\] (3.25)

\section{3.10 Proof of Theorem 3.3.1}

\textbf{Proof of Theorem 3.3.1:} Consider game \texttt{Gm}_0 given in the left panel of Fig. 3.10. By construction, it is the game \texttt{Gm}_{0, \text{idle}}(\mathcal{A}_{\text{idle}}). Next, consider game \texttt{Gm}_1, where the winning condition has been changed to checking that $(x = x')$, where $x'$ is either computed on line 8 or 9 depending on whether $w = 0$. We claim that regardless of whether $w = 0$, game \texttt{Gm}_1 returns true as long as \texttt{Gm}_0 does. Assume \texttt{Gm}_0 returns true, then $b$ is set to true. If $w = 0$, then the game \texttt{Gm}_1 sets $x'$ to $x$ at line 8, so \texttt{Gm}_1 also returns true. If $w \neq 0$, then the game \texttt{Gm}_1 computes $x'$ as per line 12 and 13. Observe that if $b$ is true, then

\[
g^2 = R_I \cdot X^{c'} .
\]
Expanding this equation using the fact that $R_i = g^{r_i,1}X^{r_i,1}$, we get

$$g^x = g^{r_i,1}X^{r_i,2} \cdot X^{c_i},$$

which means that

$$g^x = X = g^{(z-r_{i,1})w^{-1}} = g^{x'}. $$

So game $Gm_1$ must return true in this case as well. Hence

$$\Pr[Gm_0] = \Pr[Gm_1]. \quad (3.26)$$
Next, consider game $G_m_2$, which sets $x'$ differently if $w = 0$. We have

$$
Pr[G_m_1] \leq Pr[G_m_2] + Pr[G_m_2 \text{ sets bad}]
\leq Pr[G_m_2] + \frac{q}{p}.
$$

(3.27)

Above, the calculation of $Pr[G_m_2 \text{ sets bad}]$ is justified as follows. For each $C_H$ query, there is $1/p$ chance that $r_{i,2} + c_i = 0$, since $c_i$ is uniform and independent of $r_{i,2}$. Hence, the probability that there is a choice of $i$ to make $w = r_{i,2} + c_i$ zero is at most $q/p$ using the union bound. Finally, we construct adversary $A_{idl}$, given in Fig. 3.10 such that

$$
Pr[G_m_2] = \text{Adv}^{idl}_{\mathcal{G},\mathcal{g}}(A_{idl}).
$$

(3.28)

This is straight-forward, as $A_{idl}$ simulates $C_H$ and computes $x'$ exactly as $G_m_2$. 

### 3.11 Proof of Theorem 3.3.2

**Proof of Theorem 3.3.2.** Consider games $G_m_0$ given in Fig. 3.11. Game $G_m_0$ pre-samples all the $c_1, \ldots, c_q$ values at line 2, but the game behaves otherwise exactly as $G_m^{idl}_{\mathcal{G},\mathcal{g},q}(A_{idl})$. We define $Pr[G_m_0]$ to be the probability that the first component of the return value of $G_m_0$ is non-zero. Hence,

$$
Pr[G_m_0] = \text{Adv}^{idl}_{\mathcal{G},\mathcal{g},q}(A_{idl}).
$$

(3.29)

Next, consider $G_m_1$, which executes line 6 to 13 in addition to those executed by game $G_m_0$. Similar to $G_m_0$, we define $Pr[G_m_1]$ to be the probability that the first component of the return value of $G_m_1$ is non-zero. We have constructed $G_m_1$ so that it is a forked run of $G_m_0$ (with $c_1, \ldots, c_q$ viewed as inputs) as defined by the forking lemma [14]. Specifically, line 8 to 10 freshly samples challenges $c'_1, \ldots, c'_{q_2}$ after the selected forgery index $I$ before invoking $A_{idl}$ with these
### Game $G_{m_0}, G_{m_1}, G_{m_2}$

**INIT:**

1. $x \leftarrow \mathbb{Z}_p^*; X \leftarrow g^x$
2. $\rho \leftarrow \text{rand}(\mathcal{A}_{dl}) ; c_1, \ldots, c_q \leftarrow \mathbb{Z}_p$
3. $(I, z) \leftarrow \mathcal{A}_{dl}^{\text{Ctt}}(X; \rho)$
4. $b \leftarrow (g^z = R_I \cdot Y_{I^c}^z)$
5. If not $b$ then $I \leftarrow 0$
6. $G_{m_0}$: Return $(I > 0)$
7. For $i = 1, \ldots, I - 1$ do $c_i' \leftarrow c_i$
8. $c_I', c_{l+1}', \ldots, c_q' \leftarrow \mathbb{Z}_p$
9. $i \leftarrow 0 ; (I', z') \leftarrow \mathcal{A}_{dl}^{\text{Ctt}}(X; \rho)$
10. $b' \leftarrow (g^{z'} = R_{I'} \cdot Y_{I^c}^{z'}$)
11. If not $b'$ then $I' \leftarrow 0$
12. $G_{m_1}$:
   13. Return $(I = I' > 0)$ and $(c_I \neq c_I')$
13. $G_{m_2}$:
   14. If $((I \neq I')$ or $(c_I = c_I'))$ then
   15. Return $\bot$
   16. $w \leftarrow (c_I - c_I')^{-1}(z - z') \mod p$
   17. Return $(g^w = X)$

**$C_{H_1}(R)$:**

11. $i \leftarrow i + 1 ; R_i \leftarrow R$
12. Return $c_i$

**$C_{H_2}(R)$:**

13. $i \leftarrow i + 1 ; R_i' \leftarrow R$
14. Return $c_i'$

### Adversary $\mathcal{A}_{dl}(X)$:

1. $c_1, \ldots, c_q \leftarrow \mathbb{Z}_p^*; \rho \leftarrow \text{rand}(\mathcal{A}_{dl})$
2. $(I, z) \leftarrow \mathcal{A}_{dl}^{\text{Ctt}}(X; \rho)$
3. $b \leftarrow (g^z = R_I \cdot Y_{I^c}^z)$
4. If not $b'$ then Return $\bot$
5. For $i = 1, \ldots, I - 1$ do $c_i' \leftarrow c_i$
6. $c_I', c_{l+1}', \ldots, c_q' \leftarrow \mathbb{Z}_p$
7. $i \leftarrow 0 ; (I', z') \leftarrow \mathcal{A}_{dl}^{\text{Ctt}}(X; \rho)$
8. $b' \leftarrow (g^{z'} = R_{I'} \cdot Y_{I^c}^{z'}$)
9. If not $b'$ then Return $\bot$
10. If $((I \neq I')$ or $(c_I = c_I'))$ then
11. Return $\bot$
12. $w \leftarrow (c_I - c_I')^{-1}(z - z') \mod p$
13. Return $w$

**Figure 3.11:** Games $G_{m_0}, G_{m_1}, G_{m_2}$ and adversary $\mathcal{A}_{dl}$ for proof of Theorem 3.3.2

$\rho \leftarrow \text{rand}(\mathcal{A}_{dl})$ denotes sampling the random coins of $\mathcal{A}_{dl}$ and assigning it to $\rho$.

Values programmed into $C_{H_2}$. By the forking lemma, we have

$$\Pr[G_{m_0}] \leq \frac{q^2}{p} + \sqrt{q_2 \cdot \Pr[G_{m_1}]}.$$  \hspace{1cm} (3.30)
We now move onto game Gm₂, which rewrites the winning condition of Gm₁ into line 15 to 18. We claim that game Gm₂ returns true as long as game Gm₁ returns true. This is because if both flags b and b′ are true, then

\[ g^b = R_i X^{c_i} \]
\[ g^{b'} = R_{i'} X^{c_{i'}} \]

where \( i = i' > 0 \). Notice that we also have \( R_i = R_{i'} \), this is because the two runs of \( A_{idl} \) has not diverged when \( R_i \) and \( R_{i'} \) are supplied (since the first different value of \( c_{i'} \) is only supplied after \( R_{i'} \) is given). Hence, putting the two equation together, we have

\[ X^{c_i - c_{i'}} = g^{z - z'} \]

which implies the the computed value of \( w = (c_i - c_{i'})^{-1}(z - z') \) (line 17) is the correct discrete log of \( X \) base g. As a result, Gm₂ must return true as well, and

\[ \Pr[Gm_2] \geq \Pr[Gm_1] . \] (3.31)

Finally, we construct adversary \( A_{dl2} \), given in Fig. 3.11, such that

\[ \Pr[Gm_2] = \text{Adv}_{G,g}^{dl}(A_{dl}) . \] (3.32)

Adversary \( A_{dl} \) forwards its target point \( X \) to \( A_{dl} \) and simulates Gm₂, starting from line 2 of Gm₂ and ending at line 17 of Gm₂, before outputting the computed value of \( w \) as the discrete log of target point \( X \). Putting the above equations together, we obtain the claim in the theorem. □
3.12 Proof of Theorem 3.3.3

Proof of Theorem 3.3.3: We recall the convention that representation of each of the group elements $S$ and $R$ are additionally supplied when oracles $\text{NwTAR}$ and $\text{CH}$ are called. Specifically,
each of its NWTar queries must be of the form

\[ \text{NWTar}(S,(s_1,s_2)) , \]

such that \( S = g^{s_1}X^{s_2} \). And each CH query must be of the form

\[ \text{CH}(j_{\text{sel}}, R,(r_1,r_2)) , \]

such that \( R = g^{r_1}X^{r_2} \).

Consider game \( Gm_0 \) given in the left panel of Fig. 3.12. By construction, it is the game \( Gm_{\text{Xidl}}^{x_{\text{idl}}, q_1,q_2}(A_{\text{Xidl}}) \). Next, consider game \( Gm_1 \), where the winning condition has been changed to checking that \( (x = x') \), where \( x' \) is either computed on line 9 or 10 depending on whether \( w = 0 \). We claim that regardless of the value of \( w \), game \( Gm_1 \) returns true as long as \( Gm_0 \) does (\( Gm_0 \) returns the boolean value \( b \)). We check this by cases. First, if \( w = 0 \), then the game sets \( x' \) to \( x \) if \( b \) is true, so \( Gm_1 \) also returns true. If \( w \neq 0 \), then observe that if \( b \) is true, then

\[
g^z = R^1 \cdot (S^1 \cdot X^{e^1})^{c^1} .
\]

Expanding this equation using the fact that \( R = g^{r_1}X^{r_2} \) and \( S = g^{s_1}X^{s_2} \), we get

\[
g^z = g^{r_1}X^{r_2} \cdot (g^{s_1}X^{s_2} \cdot X^{e_2})^{c_1} ,
\]

which means that

\[
g^x = X = g^{(z-r_1-s_2 \cdot c_1)w^{-1}} = g^{x'} .
\]

Hence

\[
\Pr[Gm_0] = \Pr[Gm_1] .
\]
Next, consider game $G_m_2$, which sets $x'$ differently if $w = 0$. We have

$$\Pr[G_m_1] \leq \Pr[G_m_2] + \Pr[G_m_2 \text{ sets bad}]$$

$$\leq \Pr[G_m_2] + \frac{q_1 + q_2}{p}. \quad (3.34)$$

Above, the calculation of $\Pr[G_m_2 \text{ sets bad}]$ is justified as follows. First, the probability that $s_j + e_j = 0$ for any $j$ is at most $q_1/p$, since $e_j$ is uniform and independent of $s_j$. Second, assuming $s_j + e_j \neq 0$ for all $j$, then the probability that $r_i + (s_T[i] + e_T[i]) \cdot c_i = 0$ for some $i$ is at most $q_2/p$, since $c_i$ is uniform and independent of $r_i$. Finally, we construct adversary $A_{dl}$, given in the right panel of Fig. 3.12 such that

$$\Pr[G_m_2] = A_{dl}(A_{dl}). \quad (3.35)$$

This is straight-forward, as $A_{dl}$ simulates $NWTAR, CH$ and computes $x'$ exactly as $G_m_2$. \[\blacksquare\]

### 3.13 Proof of Theorem 3.3.4

**Proof of Theorem 3.3.4**: Consider games $G_m_0$ given in Fig. 3.13. Game $G_m_0$ pre-samples all the $e_j$ and $c_j$ values at line 2 and 3, but the game behaves otherwise exactly as $G_m^{xidl}_{C,g,q_1,q_2}(A_{xidl})$. We define $\Pr[G_m_0]$ to be the probability that the first component of the return value of $G_m_0$ is non-zero. Hence,

$$\Pr[G_m_0] = A_{dl}(A_{xidl}). \quad (3.36)$$

Next, consider $G_m_1$, which executes line 6 to 14 addition to those executed by game $G_m_0$. Similar to $G_m_0$, we define $\Pr[G_m_1]$ to be the probability that the first component of the return value of $G_m_1$ is non-zero. We have constructed $G_m_1$ so that it is a forked run of $G_m_0$ (with $c_1, \ldots, c_{q_2}$ viewed as inputs) as defined by the forking lemma [14]. Specifically, line 8 to 10 freshly samples
<table>
<thead>
<tr>
<th><strong>Game Gm₀, Gm₁, Gm₂</strong></th>
<th><strong>Adversary A_{idl}^{Ch}(X):</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT:</td>
<td>1. ( c₁, \ldots, c₉ \leftarrow Z_p ); ( ρ \leftarrow \text{rand}(A_{idl}) )</td>
</tr>
<tr>
<td></td>
<td>2. ((I, z) \leftarrow A_{NWTAR₁}^{ChSim}(X; ρ) )</td>
</tr>
<tr>
<td></td>
<td>3. ( b \leftarrow (g^z = R_I \cdot Y'_I) )</td>
</tr>
<tr>
<td></td>
<td>4. If ( b = \bot ) then Return (</td>
</tr>
<tr>
<td>Gm₀: Return (</td>
<td>I &gt; 0) )</td>
</tr>
<tr>
<td></td>
<td>6. For ( i, j \leftarrow 0; c'<em>i, c'</em>{i+1}, \ldots, c'_q \leftarrow Z_p )</td>
</tr>
<tr>
<td></td>
<td>7. ((I', z') \leftarrow A_{NWTAR₂}^{ChSim}(X; ρ) )</td>
</tr>
<tr>
<td></td>
<td>8. ( b' \leftarrow (g^{z'} = R_{I'} \cdot Y'_{I'}) )</td>
</tr>
<tr>
<td></td>
<td>9. If ( b' = \bot ) then Return (</td>
</tr>
<tr>
<td></td>
<td>10. If ((I \neq I') ) or ((c_I = c'_I) ) then ( j \leftarrow \text{null} )</td>
</tr>
<tr>
<td></td>
<td>11. If ( j = j + 1 ) then ( e_j \leftarrow \text{Ch}(S); S_j \leftarrow S )</td>
</tr>
<tr>
<td></td>
<td>12. ( T_j \leftarrow S_j \cdot X_e; ) Return ( e_j )</td>
</tr>
<tr>
<td></td>
<td>13. Return ( e_j )</td>
</tr>
<tr>
<td></td>
<td>14. Return ((I = I' &gt; 0) ) and ((c_I \neq c'_I) )</td>
</tr>
<tr>
<td></td>
<td>15. If ((I \neq I') ) or ((c_I = c'_I) ) then Return (</td>
</tr>
<tr>
<td></td>
<td>16. ( w \leftarrow (c_I - c'_I)^{-1}(z - z') \mod p )</td>
</tr>
<tr>
<td></td>
<td>17. Return ( (g^w = T_j) )</td>
</tr>
<tr>
<td></td>
<td>18. Return ( e_j )</td>
</tr>
<tr>
<td></td>
<td>19. Return ( e_j )</td>
</tr>
<tr>
<td></td>
<td>20. Return ( e_j )</td>
</tr>
<tr>
<td></td>
<td>21. Return ( e_j )</td>
</tr>
<tr>
<td></td>
<td>22. Return ( e_j )</td>
</tr>
<tr>
<td></td>
<td>23. Return ( e_j )</td>
</tr>
</tbody>
</table>

**Figure 3.13:** Games Gm₀, Gm₁, Gm₂ and adversary A_{idl} for proof of Theorem 3.3.4.

challenges \( c'_i, \ldots, c'_q \) after the selected forgery index \( i \) before invoking A_{xidl} with these values reprogrammed into Ch. We remark that the values of \( e₁, \ldots, e_{q₁} \), which are outputs of NWTAR.
are not resampled across the two runs of $\mathcal{A}_{\text{xidl}}$. By the forking lemma, we have

$$\Pr[\text{Gm}_0] \leq \frac{q_2}{p} + \sqrt{q_2 \cdot \Pr[\text{Gm}_1]}.$$  \hfill (3.37)

We now move onto game $\text{Gm}_2$, which rewrites the winning condition of $\text{Gm}_1$ into line 16 to 19. We claim that game $\text{Gm}_2$ returns true as long as game $\text{Gm}_1$ returns true. This is because if both flags $b$ and $b'$ are true, then

$$g^z = R_I Y_I^{c_I},$$

$$g^{z'} = R_I' Y_{I'}^{c_{I'}},$$

where $I = I' > 0$. Notice that we also have $R_I = R_I'$, this is because the two runs of $\mathcal{A}_{\text{xidl}}$ has not diverged when $R_I$ and $R_I'$ are supplied (since the first different value of $c_{\text{iforge}}$ is only supplied after $R_{I'}$ is given). Via similar reasoning, $Y_I = Y_{I'} = T_J$. Hence, putting the two equation together, we have

$$Y_i^{c_i-c_{i'}} = g^{z-z'},$$

which implies the the computed value of $w$ (line 18) is the correct discrete log of $T_J$ base $g$. As a result, $\text{Gm}_2$ must return true as well, and

$$\Pr[\text{Gm}_2] \geq \Pr[\text{Gm}_1].$$  \hfill (3.38)

Finally, we construct adversary $\mathcal{A}_{\text{idl}}$, given in Fig. 3.13 such that

$$\Pr[\text{Gm}_2] = \text{Adv}_{\text{idl}}^{\text{idl}} (\mathcal{A}_{\text{idl}}).$$  \hfill (3.39)

Crucially, in the construction of $\mathcal{A}_{\text{idl}}$, NW\text{TAR} oracle need to be simulated differently for the two runs of $\mathcal{A}_{\text{xidl}}$. In the first run, the oracle NW\text{TAR} forwards the queries to $C_H$ (that is given to our
reduction adversary from the game \( G^\text{idl}_{G_q q_1} \), while recording the responses \( e_1, \ldots, e_j \). Then, in the second run, the oracle \( \text{NwTAR}_2 \) will return previously recorded values of \( e_1, \ldots, e_j \) as long as they are available, and only starts to forward queries when it runs out of previously recorded ones. This is to simulate the behavior of \( G^2_q \), where there is one single fixed sequence of values \( e_1, \ldots, e_{q_1} \), used by the oracle \( \text{NwTAR} \). Putting the above equations together, we obtain the claim in the theorem.

3.14 Proof of Theorem 3.5.1

Proof of Theorem 3.5.1: The proof uses a game sequence. Our games will implement \( H_0, H_1 \) with lazy sampling, maintaining tables \( HF_0, HF_1 \) for this purpose. They will provide oracles \( \text{SIGN}_1, \text{SIGN}_2 \) for the first two rounds, but omit \( \text{SIGN}_3 \), since this round returns to the adversary only a quantity it could itself compute already. In \( \text{FIN} \) (for example Figure 3.14) we assume the query is non-trivial, meaning lines 6,7 of Figure 3.4 return true, and these lines are thus omitted. We start with games \( G_{m_0}, G_{m_1} \) in Figure 3.14. Game \( G_{m_0} \) includes the boxed code, and we claim that

\[
\text{Adv}_{\text{MS-uf}}(A) = \Pr[G_{m_0}(A)].
\] (3.40)

Let us explain. We wish to move to a game where signing queries are answered without using the secret key \( sk \). Naturally, we expect, for this, to use the zero-knowledge property of the Schnorr scheme. But certain obstacles must be removed before we can do this, and this will take a few steps. The first obstacle we address is that the BN-commitment \( t_{u,k} = H_0((k, R_{u,k})) \) may leak information about \( R_{u,k} \). Rather than define \( t_{u,k} \) in this way, games \( G_{m_0}, G_{m_1} \) accordingly pick it at random at line 3. The reason for the boxed code at line 4 is that, under the “true” assignment \( t_{u,k} = H_0((k, R_{u,k})) \), having \( R_{u,k_u} = R_{u',k_{u'}} \) would imply \( t_{u,k_u} = t_{u',k_{u'}} \). At line 8, now
INIT:  / Games Gm₀–Gm₇
1   (pk, sk) ← MS.Kg ; Return pk
2 NS(k, pk, m):  / Games Gm₀, Gm₁
3   u ← u + 1 ; kᵦ ← k ; pk ← pk ; pk_u ← pk ; m_u ← m ; n_u ← pk
4   CommitStage_u ← true ; r_u,k ← s Z_p : R_u,k ← g^{r_u,k} ; t_u,k ← s \{0,1\}^ℓ
5   If (∃ u' < u : R_u,k = R_u',k) then bad ← true ; \{R_u,k ← t_u',k\}
6   If (HF₀[(k, R_u,1)] \neq ⊥) then bad ← true ; \{R_u,k ← HF₀[(k, R_u,1)]\}
7   Return R_u,k
8 SIGN₀(s,t):  / Games Gm₀, Gm₁
9   k ← kᵦ ; t[k] ← tᵦ,k ; tᵦ ← t ; CommitStageᵦ ← false
10  HF₀[(k, Rᵦ,k)] ← tᵦ,k ; Return Rᵦ,k
11 SIGN₁(s,R):  / Games Gm₀, Gm₁, Gm₂
12   k ← kᵦ ; R[k] ← Rᵦ,k
13   For i=1,...,n do yi ← H₀((i, R[i]))
14   If (∃ i : yᵦ \neq tᵦ[i]) then Return ⊥
15   Rᵦ ← \prod_{i=1}^{n} R[i] ; cᵦ,k ← H₁((k, Rᵦ, pkᵦ, mᵦ)) ; zᵦ,k ← sk \cdot cᵦ,k + rᵦ,k
16   Return zᵦ,k
17 H₀(x):  / Games Gm₀, Gm₁
18   If (HF₀[x] \neq ⊥) then Return HF₀[x]
19   HF₀[x] ← s \{0,1\}^ℓ
20   If (∃ u' : x = (k_u', R_u',k) and CommitStage_u') then
21      bad ← true ; \{HF₀[x] ← t_u',k\}
22   Return HF₀[x]
23 H₁(x):  / Games Gm₀–Gm₇
24   If (HF₁[x] \neq ⊥) then Return HF₁[x]
25   HF₁[x] ← s Z_p ; Return HF₁[x]
26 FIN(pk, m, (R,z)):  / Games Gm₀–Gm₇
27   n ← pk
28   For i=1,...,n do ci ← H₁((i, R, pk, m))
29   X ← \prod_{i=1}^{n} pk[i]^{cᵦ} ; Return (g^X = RX)

Figure 3.14: Games Gm₀, Gm₁ for proof of Theorem 3.5.1 Some procedures will be included in later games, as indicated. A box around the name of a game following an oracle means the boxed code in that oracle is included in the game.
that the BN-commitments $t$ of all players are known, the games ensure that $t_{u,k}$ indeed equals $H_0((k,R_{u,k}))$. This is consistent with the real game only if the hash function was not already defined at this point, captured by setting bad at line 17. The boolean CommitStage ensures that bad is only set prior to the release of $R_{s,k}$, since the adversary can set it with probability one if it knows $R_{s,k}$. This justifies Eq. (3.40).

Games $Gm_0, Gm_1$ are identical-until-bad, so by the Fundamental Lemma of Game Playing [19]

$$\Pr[Gm_0(\mathcal{A})] \leq \Pr[Gm_1(\mathcal{A})] + \Pr[Gm_1(\mathcal{A}) \text{ sets bad}] .$$

The probability of setting bad at line 4 is at most $\left(0 + 1 + \cdots + q_s - 1\right)/p$, and the probabilities of setting bad at line 5 and line 17 are at most $q_s q_0/p$, so

$$\Pr[Gm_1(\mathcal{A}) \text{ sets bad}] \leq \frac{q_s(q_s - 1)}{2p} + \frac{2q_s q_0}{p} = \frac{q_s(4q_0 + q_s - 1)}{2p} .$$

Game $Gm_2$ changes the $NS, SIGN_0, H_0$ oracles as shown in Figure 3.15, maintaining the other oracles of $Gm_1$ from Figure 3.14. It drops redundant code, which allows it to move the choice of $R_{s,k}$ to line 28. At line 40, it also introduces a table HI to maintain an inverse of the hash function, but does not use this. We have

$$\Pr[Gm_1(\mathcal{A})] = \Pr[Gm_2(\mathcal{A})] .$$

Game $Gm_3$ (oracles shown across Figures 3.15 and 3.14) aims to figure out the $R_{s,j}$-values of parties $j \neq k$ before having to supply $R_{s,k}$, because we will later need these to program $H_1$ values. It does this by “inverting” the BN-commitments, meaning at line 30 it seeks inputs to $H_0$ that result in the BN-commitments in $t$. If these cannot be found, then random values are chosen instead at line 31. (Not finding the inverses is not yet a bad event. It can happen with high
probability. It becomes a bad event only at line 36 when the BN-commitments are verified.) The computation of $t$ at that line is only to ensure that $H_0$ has been called; this variable will not be used. These steps do not change what the oracles return compared to Gm2, so we have

$$\Pr[\text{Gm}_2(\mathcal{A})] = \Pr[\text{Gm}_3(\mathcal{A})].$$

Moving to game Gm4, the change is only at line 36, which now includes the boxed code. The hope here is that the $R_s^*$ obtained at lines 30,31 is correct with high probability. The boxed code

\begin{figure}[h]

\begin{verbatim}
NS(k, pk, m): Gm Gm \n24 u ← u + 1 ; k_u ← k : pk[1] ← pk : pk_u ← pk : m_u ← m : n_u ← |pk|
25 t_{u, 1} ← \{0, 1\}^\ell ; Return t_{u, 1}

SIGN_0(s, t): Gm
26 t[1] ← t_{u, 1} ; t_s ← t ; r_s, 1 ← \mathbb{Z}_p ; R_{s, 1} ← g^{r_s, 1} ; HF_0[(1, R_{s, 1})] ← t_{s, 1}
27 Return R_{s, 1}

SIGN_0(s, t): Gm
28 k ← k_s ; t[k] ← t_{s, k} ; t_s ← t ; r_{s, k} ← \mathbb{Z}_p ; R_{s, k} ← g^{r_{s, k}} ; HF_0[(k, R_{s, k})] ← t_{s, k}
29 For i = 1, ..., n do
30 If (H_0[i, t_s[i]] \neq ⊥) then R_s[i] ← H_0[i, t_s[i]]
31 Else R_s[i] ← ⊥ ; t ← H_0((i, R_s[i]))
32 Return R_{s, k}

SIGN_1(s, R): Gm
33 k ← k_s ; R[k] ← R_{s, k}
34 For i = 1, ..., n do y_i ← H_0((i, R[i]))
35 If (\exists i : y_i \neq t_s[i]) then Return ⊥
36 If (R \neq R_s^*) then bad ← true ; [R \leftarrow R_s^*]
37 R_s ← \prod_{i=1}^n R[i] ; c_{s, k} ← H_1((k, R_s, pk_s, m_s)) ; z_{s, k} ← sk \cdot c_{s, k} + r_{s, k}
38 Return z_{s, k}

H_0(x): Gm
39 If (HF_0[x] \neq ⊥) then Return HF_0[x]
40 HF_0[x] ← \{0, 1\}^\ell ; (i, R) ← x ; H_0[i, HF_0[x]] ← R ; Return HF_0[x]
\end{verbatim}

\end{figure}
Figure 3.16: Games for proof of Theorem 3.5.1

equarizes that in Gm₄, it is always correct. Since Gm₃, Gm₄ are identical-until-bad we have

\[ \Pr[\text{Gm₃}(A)] \leq \Pr[\text{Gm₄}(A)] + \Pr[\text{Gm₃}(A) \text{ sets bad}] . \]

Line 36 can only set bad if \( y_i = t_s[i] \) for all \( i \), due to line 35. So it is set only if there is a collision in \( H_0 \)-values, or no query hashing to \( t_s[i] \) was made prior to the latter being provided, but is made later. Thus

\[ \Pr[\text{Gm₃}(A) \text{ sets bad}] \leq \frac{q_0^2 + nq_0}{2^\ell} . \quad (3.41) \]

In game Gm₄, the \( R \) queried to \( \text{SIGN}_1 \) is the same as the \( R^* \) determined in \( \text{SIGN}_0 \), allowing
game $G_{m_5}$ (Figure 3.16) to move line 37 into $\text{SIGN}_0$ as line 45 and to simplify $\text{SIGN}_1$. We have

$$\Pr[G_{m_5}(A)] = \Pr[G_{m_5}(A)].$$

Now that $R_s$ is determined prior to the release of $R_{s,k_s}$, it becomes possible to successfully program $H_1$ via the zero-knowledge simulation. Game $G_{m_6}$ of Figure 3.16 does this, setting $\text{bad}$ at line 56 if the programming was precluded by the hash value already being defined, and including the boxed code to correct. We have

$$\Pr[G_{m_5}(A)] = \Pr[G_{m_6}(A)].$$

Games $G_{m_6}, G_{m_7}$ (Figure 3.16) are identical-until-$\text{bad}$, so

$$\Pr[G_{m_6}(A)] \leq \Pr[G_{m_7}(A)] + \Pr[G_{m_7}(A) \text{ sets } \text{bad}]. \quad (3.42)$$

When line 56 is executed, the adversary has as yet no information about $R_s$, which means

$$\Pr[G_{m_7}(A) \text{ sets } \text{bad}] \leq \frac{q_s q_1}{p}. \quad (3.43)$$

We now build an adversary $A_{\text{idl}}$ so that

$$\text{Adv}_{U.G.q}^{A_{\text{idl}}}(A_{\text{idl}}) \geq \Pr[G_{m_7}(A_{m_s})]. \quad (3.44)$$

We specify $A_{\text{idl}}$ in Figure 3.17. It forwards the public key $pk$ to $A_{m_s}$. Simulating signatures without knowing the secret key, as $A_{\text{idl}}$ needs to do, is now easy because the oracles of games $G_{m_7}$ already did this, and $A_{\text{idl}}$ can just use the same code. Line 17 to 19 programs the challenge $c_k$ of the target public key by first deriving commitment $R_k$, which is then submitted to $\text{Ct}_k$ to derive $c_k$. Since $G_{m_7}^{A_{\text{idl}}} \text{ game also samples the challenge uniformly at random, this does not
change the behavior of $H_1$. However, if a forgery $(pk, m, (R, z))$, then it must be that

$$g^z = R \cdot \prod_{i=1}^{pk} pk[i]^{H_1(i, R, pk, m)} = R_{j,k} \cdot pk^{c_{j,k}}.$$  

So $A_{idl}$ wins game $G_{idl}^{M, G, S, q}$. Eq. (3.3) is obtained by putting the above all together. 

**Figure 3.17:** Adversary $A_{idl}$ for Theorem 3.5.1
3.15 Proof of Theorem 3.6.1

Let \( G \) be a group of prime order \( p \) with generator \( g \). Let \( MS = \text{MuSig} [G, g, \ell] \) be the associated MuSig multi-signature scheme. Let \( \mathcal{A}_{ms} \) be an adversary for game \( G_{MS}^{ms-uf} \) of Figure 3.4. We shall fix these quantities for the rest of the proof. The first lemma relates the advantage of \( \mathcal{A}_{ms} \) against \( G_{MS}^{ms-uf} \) to a simplified game \( G_{ms}^{simp} \) (given in Fig. 3.18).

**Lemma 3.15.1** Assume the execution of game \( G_{MS}^{ms-uf} \) with \( \mathcal{A}_{ms} \) has at most \( q_0, q_1, q_2, q_s \) distinct queries to \( H_0, H_1, H_2, \text{NS} \), respectively, and the number of parties (length of verification-key vector) in queries to \( \text{NS} \) and \( \text{FIN} \) is at most \( n \). Let \( \alpha = q_s(4q_0 + 2q_1 + q_s) + 2q_1 q_2 \) and \( \beta = q_0(q_0 + n) \). Then,

\[
\text{Adv}_{MS}^{ms-uf}(\mathcal{A}_{ms}) \leq \text{Pr}[G_{ms}^{simp}(\mathcal{A}_{ms})] + \frac{\alpha}{2p} + \frac{\beta}{2\ell} \, .
\]

(3.45)

The second lemma constructs the reduction adversary against \( G_{G, g, q_2, q_1}^{xidl} \).

**Lemma 3.15.2** Assume the execution of game \( G_{MS}^{ms-uf} \) with \( \mathcal{A}_{ms} \) has at most \( q_0, q_1, q_2, q_s \) distinct queries to \( H_0, H_1, H_2, \text{NS} \), respectively. We construct an adversary \( \mathcal{A}_{xidl} \) for game \( G_{G, g, q_2, q_1}^{xidl} \) (shown explicitly in Figure 3.23) such that

\[
\text{Pr}[G_{ms}^{simp}(\mathcal{A}_{ms})] \leq \text{Adv}_{G, g, q_2, q_1}^{xidl}(\mathcal{A}_{xidl}) \, .
\]

(3.46)

**Proof of Lemma 3.15.1**:

The proof uses a game sequence. Our games will implement \( H_0, H_1, H_2 \) with lazy sampling, maintaining tables HF0, HF1, HF2 for this purpose. They will provide oracles \( \text{SIGN}_0, \text{SIGN}_1 \) while omitting \( \text{SIGN}_2 \), since this round returns to the adversary only a quantity it could itself compute already. In \( \text{FIN} \) (for example Figure 3.19) we assume the query is non-trivial, meaning lines 6,7 of Figure 3.4 return true, and these lines are thus omitted. We start with games \( G_{m0}, G_{m1} \),
INIT:
1. \((pk, sk) \leftarrow MS.Kg\); Return \(pk\)

NS\((k, pk, m)\):
2. \(u \leftarrow u + 1; k_u \leftarrow k; pk[1] \leftarrow pk; pk_u \leftarrow pk; m_u \leftarrow m; n_u \leftarrow |pk|\)
3. \(t_{u, 1} \leftarrow \{0, 1\}^f\); Return \(t_{u, 1}\)

SIGN\(_1\)(\(x, R\)):
4. \(k \leftarrow k_s; R[k] \leftarrow R_{x, k}\)
5. For \(i = 1, \ldots, n_s\) do \(y_i \leftarrow H_0 ((i, R[i]))\)
6. If \((\exists i : y_i \neq T_s[i])\) then Return \(\bot\) else Return \(z_{s, k}\)

SIGN\(_0\)(\(x, t\)):
7. \(k \leftarrow k_s; t[k] \leftarrow t_{s, k}; t_s \leftarrow t\)
8. \(c_{s, k} \leftarrow Z_p; z_{s, k} \leftarrow Z_p; R_{s, k} = g^{s_z} pk^{-c_{s, k}}; HF_0 [(k, R_{s, k})] \leftarrow t_{s, k}\)
9. For \(i = 1, \ldots, n_s\) do
10. If \((HF_0[i, t_s[i]] \neq \bot)\) then \(R_s[i] \leftarrow HF_0[i, t_s[i]]\)
11. Else \(R_s[i] \leftarrow \emptyset; t \leftarrow HF_0 ((i, R_s[i]))\)
12. \(R_s \leftarrow \prod_{i=1}^{n_s} R_s[i]\)
13. \(HF_1 [(k, R_s, pk_s, m_s)] \leftarrow c_{s, k}\); Return \(R_{s, k}\)

\(H_0(x)\):
14. If \((HF_0[x] \neq \bot)\) then Return \(HF_0[x]\)
15. \(HF_0[x] \leftarrow \{0, 1\}^f; (i, R) \leftarrow x; HF_0 [i, HF_0[x]] \leftarrow R\); Return \(HF_0[x]\)

\(H_1(x)\):
16. If \((HF_1[x] \neq \bot)\) then Return \(HF_1[x]\)
17. \((R, apk, m) \leftarrow x; TV[apk] \leftarrow TV[apk] \cup \{x\}\)
18. \(HF_1 [x] \leftarrow Z_p\); Return \(HF_1 [x]\)

\(H_2(x)\):
19. If \((HF_2[x] \neq \bot)\) then Return \(HF_2[x]\)
20. \((k, pk) \leftarrow x; \\text{For } i = 1, \ldots, |pk| \text{ do } HF_2 [(i, pk)] \leftarrow e_i \leftarrow Z_p\)
21. \(apk \leftarrow \prod_{i=1}^{|pk|} pk[i]^s; \\text{For } y \in TV[apk] \text{ do } HF_1 [y] \leftarrow \bot\)
22. Return \(HF_2[x]\)

FIN\((pk, m, (R, z))\):
23. For \(i = 1, \ldots, |pk|\) do \(c_i \leftarrow H_1 ((i, R, pk, m)); e_i \leftarrow H_2 ((i, pk))\)
24. \(X \leftarrow \prod_{i=1}^{|pk|} pk[i]^{c_i}; \text{Return } (g^z = RX)\)

Figure 3.18: Game \(G_{simp}\) for proof of Theorem 3.6.1
INIT: $I$ Games $G_{m_0} - G_{m_9}$
1 $(pk, sk) \leftarrow M.S.K_g$; Return $pk$

NS$(k, pk, m)$: $I$ Games $G_{m_0}$, $G_{m_1}$
2 $u \leftarrow u + 1$; $k_u \leftarrow k$; $pk[k] \leftarrow pk$; $pk_u \leftarrow pk$
3 $m_u \leftarrow m$; $n_u \leftarrow |pk|$; CommitStage$_n \leftarrow$ true
4 $r_{u,k} \leftarrow Z_p$; $R_{u,k} \leftarrow g^{r_{u,k}}$; $t_{u,k} \leftarrow s \{0,1\}^\ell$
5 If $(\exists u' < u : R_{u,k} = R_{u',k'})$ then bad$ \leftarrow$ true; $t_{u,k} \leftarrow t_{u',k'}$
6 If $(HF_0(k, R_{u,k}) \neq \bot)$ then bad$ \leftarrow$ true; $t_{u,k} \leftarrow HF_0(k, R_{u,k})$
7 Return $t_{u,k}$

SIGN$_0(s, t)$: $I$ Games $G_{m_0}$, $G_{m_1}$
8 $t[k] \leftarrow t_{s,k}$; $t \leftarrow t$; CommitStage$_s \leftarrow$ false
9 $HF_0[(k, R_{s,k})] \leftarrow t_{s,k}$; Return $R_{s,k}$

SIGN$_1(s, R)$: $I$ Games $G_{m_0}$, $G_{m_1}$, $G_{m_2}$
10 $R[k] \leftarrow R_{s,k}$
11 For $i = 1, \ldots, n_s$ do $y_i \leftarrow H_0((i, R(i))$
12 If $(\exists i : y_i \neq t_{s,i})$ then Return $\bot$
13 $R_s \leftarrow \prod_{i=1}^{n_s} R[i]$; $c_{s,k} \leftarrow H_1((k, R_s, pk_s, m_s))$; $z_{s,k} \leftarrow sk \cdot c_{s,k} + r_{s,k}$
14 Return $z_{s,k}$

$H_0(x)$: $I$ Games $G_{m_0}$, $G_{m_1}$
15 If $(HF_0[x] \neq \bot)$ then Return $HF_0[x]$
16 $HF_0[x] \leftarrow s \{0,1\}^\ell$; If $(\exists u' : x = (k_{u'}, R_{u',k'}))$ and CommitStage$_{u'}$ then
17 bad$ \leftarrow$ true; $HF_0[x] \leftarrow t_{u',k'}$
18 Return $HF_0[x]$

$H_1(x)$: $I$ Games $G_{m_0} - G_{m_7}$
19 If $(HF_1[x] \neq \bot)$ then Return $HF_1[x]$
20 $HF_1[x] \leftarrow s Z_p$; Return $HF_1[x]$

$H_2(x)$: $I$ Games $G_{m_0} - G_{m_7}$
21 If $(HF_2[x] \neq \bot)$ then Return $HF_1[x]$
22 $HF_1[x] \leftarrow s Z_p$; Return $HF_1[x]$

FIN$(k, pk, m, (R, z))$: $I$ Games $G_{m_0} - G_{m_9}$
23 For $i = 1, \ldots, |pk|$ do $c_i \leftarrow H_1((i, R, pk, m))$; $e_i \leftarrow H_2((i, pk))$
24 $X \leftarrow \prod_{i=1}^{pk} pk[i]^{c_i \cdot e_i}$; Return $(g^X = RX)$

**Figure 3.19:** Games $G_{m_0}, G_{m_1}$ for proof of Theorem 3.6.1 Some procedures will be included in later games, as indicated. A box around the name of a game following an oracle means the boxed code in that oracle is included in the game.
in Figure 3.19, Game Gm₀ includes the boxed code, and we claim that

\[ \text{Adv}^{\text{ms uf}}(\mathcal{A}) = \Pr[\text{Gm}_0(\mathcal{A})] . \]  

(3.47)

Games Gm₀, Gm₁ are identical-until-bad, so by the Fundamental Lemma of Game Playing [19]

\[ \Pr[\text{Gm}_0(\mathcal{A})] \leq \Pr[\text{Gm}_1(\mathcal{A})] + \Pr[\text{Gm}_1(\mathcal{A}) \text{ sets bad}] . \]

The probability of setting \text{bad} at line 4 is at most \((0 + 1 + \cdots + q_s - 1)/p\), while the probabilities of setting it at line 5 and 15 are at most \(q_s q_0/p\) so

\[ \Pr[\text{Gm}_1(\mathcal{A}) \text{ sets bad}] \leq \frac{q_s(q_s - 1)}{2p} + 2 \cdot \frac{q_s q_0}{p} = \frac{q_s(4q_0 + q_s - 1)}{2p} . \]

Game Gm₂ changes the NS, SIGN₀, H₀ oracles as shown in Figure 3.20, maintaining the other oracles of Gm₁ from Figure 3.19. It drops redundant code, which allows it to move the choice of R_s₁ to line 29. At line 31, it also introduces a table HI to maintain an inverse of the hash function, but does not yet use this. We have

\[ \Pr[\text{Gm}_1(\mathcal{A})] = \Pr[\text{Gm}_2(\mathcal{A})] . \]

Game Gm₃ (oracles shown across Figures 3.20 and 3.19) aims to figure out the R_s,j-values of parties \(j \neq k\) before having to supply R_s,k, because we will later need these to program H₁ values. It does this by “inverting” the BN-commitments, meaning at line 27 it seeks inputs to H₀ that result in the BN-commitments in \(t\). If these cannot be found, then random values are chosen instead at line 37. (Not finding the inverses is not yet a bad event. It can happen with high probability. It becomes a bad event only at line 37 when the BN-commitments are verified.) The computation of \(t\) at that line is only to ensure that H₀ has been called; this variable will not be
NS($k, pk, m$): / Games Gm$_2$–Gm$_9$

25 $u \leftarrow u+1$ ; $k_u \leftarrow k$ ; $pk[.] \leftarrow pk$ ; $pk_u \leftarrow pk$ ; $m_u \leftarrow m$ ; $n_u \leftarrow |pk|$

26 $t_{u,k} \leftarrow \{0, 1\}^\ell$ ; Return $t_{u,k}$

SIGN$_0(s, t)$: / Game Gm$_2$

27 $k \leftarrow k_s$ ; $t[k] \leftarrow t_{s,k}$ ; $t_s \leftarrow t$ ; $r_{s,k} \leftarrow Z_p$ ; $R_{s,k} \leftarrow g^{r_{s,k}}$ ; $HF_0[(k, R_{s,k})] \leftarrow t_{s,k}$

28 Return $R_{s,k}$

SIGN$_0(s, t)$: / Games Gm$_3$, Gm$_4$

29 $k \leftarrow k_s$ ; $t[k] \leftarrow t_{s,k}$ ; $t_s \leftarrow t$ ; $r_{s,k} \leftarrow Z_p$ ; $R_{s,k} \leftarrow g^{r_{s,k}}$ ; $HF_0[(k, R_{s,k})] \leftarrow t_{s,k}$

30 For $i = 1, \ldots, n$, do

31 If $(HF_0[i, t_s[i]] \neq \bot)$ then $R'_s[i] \leftarrow HF_0[i, t_s[i]]$

32 Else $R'_s[i] \leftarrow \emptyset$ ; $t \leftarrow H_0((i, R'_s[i]))$

33 Return $R_{s,k}$

SIGN$_1(s, R)$: / Games Gm$_3$, Gm$_4$

34 $R[k] \leftarrow R_{s,k}$

35 For $i = 1, \ldots, n$, do $y_i \leftarrow H_0((i, R[i]))$

36 If $(\exists i : y_i \neq t_s[i])$ then Return $\bot$

37 If $(R \neq R'_s)$ then bad $\leftarrow$ true ; $[R \leftarrow R'_s]$

38 $R_s \leftarrow \prod_{i=1}^n R[i]$ ; $c_{s,k} \leftarrow H_1((k, R_s, pk_s, m_s))$ ; $z_{s,k} \leftarrow sk \cdot c_{s,k} + r_{s,k}$

39 Return $z_{s,k}$

$H_0(x)$: / Games Gm$_2$–Gm$_9$

40 If $(HF_0[x] \neq \bot)$ then Return $HF_0[x]$

41 $HF_0[x] \leftarrow \{0, 1\}^\ell$ ; $(i, R) \leftarrow x$ ; $HF_0[i, HF_0[x]] \leftarrow R$ ; Return $HF_0[x]$

Figure 3.20: Games for proof of Theorem 3.6.1

used. These steps do not change what the oracles return compared to Gm$_2$, so we have

$$\Pr[\text{Gm}_2(\mathcal{A})] = \Pr[\text{Gm}_3(\mathcal{A})].$$

Moving to game Gm$_4$, the change is only at line 33, which now includes the boxed code. The hope here is that the $R'_s$ obtained at lines 32,33 is correct with high probability. The boxed code ensures that in Gm$_4$, it is always correct. Since Gm$_3$, Gm$_4$ are identical-until-bad we have

$$\Pr[\text{Gm}_3(\mathcal{A})] \leq \Pr[\text{Gm}_4(\mathcal{A})] + \Pr[\text{Gm}_3(\mathcal{A}) \text{ sets bad}].$$

168
Line 38 can only set bad if \( y_i = t_s[i] \) for all \( i \), due to line 37. So it is set only if there is a collision in \( H_0 \)-values, or no query hashing to \( t_s[i] \) was made prior to the latter being provided, but is made later. Thus

\[
\Pr[\text{Gm}_3(\mathcal{A}) \text{ sets bad}] \leq \frac{q_0^2 + nq_0}{2^s}. \tag{3.48}
\]

In game \( \text{Gm}_4 \), the \( R \) queried to \( \text{SIGN}_1 \) is the same as the \( R^* \) determined in \( \text{SIGN}_0 \), allowing game \( \text{Gm}_5 \) (Figure 3.21) to move line 38 into \( \text{SIGN}_0 \) as line 46 and to simplify \( \text{SIGN}_1 \). We have

\[
\Pr[\text{Gm}_4(\mathcal{A})] = \Pr[\text{Gm}_5(\mathcal{A})].
\]
Now that $R_s$ is determined prior to the release of $R_{s,k_4}$, it becomes possible to successfully program $H_1$ via the zero-knowledge simulation. Game $G_{m6}$ of Figure 3.21 does this, setting $bad$ at line 57 if the programming was precluded by the hash value already being defined, and including the boxed code to correct. We have

$$Pr[G_{m5}(\mathcal{A})] = Pr[G_{m6}(\mathcal{A})].$$

Games $G_{m6}, G_{m7}$ (Figure 3.21) are identical-until-bad, so

$$Pr[G_{m6}(\mathcal{A})] \leq Pr[G_{m7}(\mathcal{A})] + Pr[G_{m7}(\mathcal{A}) \text{ sets } bad].$$  \hspace{1cm} (3.49)

When line 57 is executed, the adversary has as yet no information about $R_s$, which means

$$Pr[G_{m7}(\mathcal{A}) \text{ sets } bad] \leq \frac{q_s q_1}{p}. \hspace{1cm} (3.50)$$

Moving on, let us consider games $G_{m8}$ and $G_{m9}$ in Fig. 3.22 which differ from $G_{m7}$ in modifications to oracles $H_1$ and $H_2$. Oracle $H_1$ now keeps track of a table $TV$, that stores for each aggregate key $apk$ the set of $H_1$ queries that contain it. It otherwise behave identically to $G_{m7}.H_1$.

---

**Figure 3.22:** Games for proof of Theorem 3.6.1

<table>
<thead>
<tr>
<th>H_1(x)</th>
<th>/ Game G_{m8}, G_{m9}</th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td>If (HF_1[x] \neq \bot) then Return HF_1[x]</td>
</tr>
<tr>
<td>60</td>
<td>(R, apk, m) \leftarrow x; TV[apk] \leftarrow TV[apk] \cup {x}</td>
</tr>
<tr>
<td>61</td>
<td>HF_1[x] \leftarrow \mathbb{Z}_p; Return HF_1[x]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>H_2(x)</th>
<th>/ Game G_{m8}, G_{m9}</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>If (HF_2[x] \neq \bot) then Return HF_2[x]</td>
</tr>
<tr>
<td>63</td>
<td>(\cdot, pk) \leftarrow x; For i = 1, \ldots,</td>
</tr>
<tr>
<td>64</td>
<td>apk \leftarrow \prod_{i=1}^{</td>
</tr>
<tr>
<td>65</td>
<td>If TV[apk] \neq \bot then</td>
</tr>
<tr>
<td>66</td>
<td>bad \leftarrow true; For y \in TV[apk] do HF_2[y] \leftarrow \bot</td>
</tr>
<tr>
<td>67</td>
<td>Return HF_2[x]</td>
</tr>
</tbody>
</table>
Adversary $A_{\text{sidl}}^{C_{\text{H}}}(pk)$:

1. $(pk,m,(R,z)) \leftarrow A^{NS,\text{SIGN}_0,\text{SIGN}_1,H_0,H_2}(pk)$
2. $apk \leftarrow \prod_{i=1}^{\lceil \log |pk| \rceil} H_2((i, pk))$; Return $(T[\{apk,R,m\}], z)$

$H_1(x)$:
3. If $(H_1[x] \neq \bot)$ then Return $H_1[x]$
4. $(R,apk,m) \leftarrow x$; $TV[apk] \leftarrow TV \cup \{x\}$
5. If $(TJ[apk] = \bot)$ then Return $H_1[x] \leftarrow \mathbb{Z}_p$
6. $t \leftarrow t + 1$; $TI[x] \leftarrow t$
7. $H_1[x] \leftarrow c_t \leftarrow \text{Ch}(TJ[apk], R)$; Return $H_1[x]$

$H_2(x)$:
8. If $(H_2[x] \neq \bot)$ then Return $H_2[x]$
9. $(\cdot, pk) \leftarrow x$; If $(pk \notin pk)$ then Return $H_2[x] \leftarrow \mathbb{Z}_p$
10. $j \leftarrow j + 1$; $k \leftarrow \minInd(pk)$; If $(x \neq (k, pk))$ then Return $H_2[x] \leftarrow \mathbb{Z}_p$
11. $S \leftarrow \prod_{i \neq k} H_2((i, pk))$
12. $H_2[x] \leftarrow e_j \leftarrow \text{NW\text{TAR}}(S)$; $apk \leftarrow S \cdot pk^e_j$; $TJ[apk] \leftarrow j$
13. For $y \in TV[apk]$ do $H_1[y] \leftarrow \bot$
14. Return $H_2[x]$

Figure 3.23: Adversary $A_{\text{sidl}}$ for Theorem 3.6.1. Oracles $NS, \text{SIGN}_0, \text{SIGN}_1, H_0$ are copied from game $G_{\text{m simp}}$ (Fig. 3.18).

Oracle $G_{m8}.H_2$ does not contain the boxed code, which makes the oracle behave identically to $G_{m7}.H_2$. So, we have

$$\Pr[G_{m7}(A)] = \Pr[G_{m8}(A)] \quad (3.51)$$

By construction, $G_{m7}$ and $G_{m8}$ are identical-until-bad, hence

$$\Pr[G_{m8}(A)] \leq \Pr[G_{m9}(A)] + \Pr[G_{m8} \text{ sets bad}] \quad (3.52)$$

$$\leq \Pr[G_{m9}(A)] + \frac{q_1q_2}{p}, \quad (3.53)$$

where the last inequality is by the fact that each $H_2$ query has probability at most $q_1/p$ of setting bad. Lastly, we note that $G_{m9}$ and $G_{m \text{simp}}$ are identical. This completes the proof of Lemma 3.15.1.
**Proof of Lemma 3.15.2** Consider $A_{\text{xidl}}$ in Figure 3.23. It forwards the public key $pk$ to $A_{\text{ms}}$. Simulating signatures without knowing the secret key can be done exactly as $G_{\text{ms}}^{\text{simp}}$. To break $G_{G,q_2,q_1}^{\text{xidl}}$, our adversary $A_{\text{xidl}}$ needs to program $H_1$ and $H_2$. For each $H_2$ query, Line 10 to 12 programs the response $e_j$ for the target public key by first deriving commitment $S = \prod_{i \neq k} pk[i]^{e_i}$, which is then submitted to $\text{NW\&TAR}$ to derive $e_k$ that is returned as the response. By construction, the corresponding aggregate public key $apk = S \cdot pk^{e_k}$ is exactly the target $T_j$ recorded by $G_{G,q_2,q_1}^{\text{xidl}}$ for this $\text{NW\&TAR}$ query. For each $H_1$ query, our adversary first uses the aggregate public key $apk$ find the corresponding $H_2$ query via table $TJ$. If possible, then the adversary proceeds to program in a challenge using the challenge oracle $C_H$ of XIDL. If this is not possible, the adversary simply simulates $H_1$ honestly. If a forgery $(pk,m,(R,z))$ is valid, then it must be that

$$g^z = R \cdot \prod_{i=1}^{pk} apk^{H_1((R,apk,m))},$$

where $apk = \prod_{i=1}^{pk} pk[i]^{H_2((i,pk))}$. Observe that call involving a fresh vector $pk$ to oracle $H_2$ erases the table $HF_1$ at every entry associated with the derived $apk$. Hence, our adversary can use the above relation to directly break XIDL. In other words, the value of $z$ included in the forgery makes the following equation true in game $G_{G,q_2,q_1}^{\text{xidl}}$, $g^z = R \cdot T_j^{e_i}$, where $j = TJ[apk]$ and $i = TI[(R,apk,m)]$. This justifies Equation (3.46).

---

**3.16 Proof of Theorem 3.7.1**

The first step in the proof is to move from the security game $G_{\text{MS}}^{\text{ms-uf}}$ to a game where the signing oracles can be simulated without the target secret key. We encapsulate this in the lemma below, which works strictly in the standard model, meaning it does not require adversaries involved to be algebraic. This allows our latter standard model proof of security for $\text{HBMS}$ to also rely on this lemma.
Lemma 3.16.1 Let \( G \) be a group of prime order \( p \) with generator \( g \). Let \( MS = HBMS[G, g] \) be the scheme specified in Fig. 3.8. Let \( A_{ms} \) be an adversary for game \( G_{ms}^{ms-uf} \) of Fig. 3.4. Assume the execution of game \( G_{ms}^{ms-uf} \) with \( A_{ms} \) has at most \( q_0, q_1, q_2 \) distinct queries to \( H_0, H_1, H_2 \) respectively. Let \( \rho \in [0, 1] \) be a real number. Consider games \( G_{m0} \) and \( G_{m1, \rho} \) give in Fig. 3.24. Then,

\[
\text{Adv}_{MS}^{ms-uf}(A_{ms}) = \Pr[G_{m0}(A_{ms})]
\]

\[
= \Pr[G_{m1, \rho}(A_{ms}) | \text{Gm}_{1, \rho}(A_{ms}) \text{ does not abort }].
\]

Moreover, the probability that game \( G_{m1} \) does not abort is

\[
\Pr[G_{m1, \rho}(A_{ms}) \text{ does not abort}] = \rho^{q_0},
\]

which is 1 if \( \rho = 1 \).

Proof of Lemma 3.16.1: Consider games \( G_{m0} \) and \( G_{m1, \rho} \) given in Fig. 3.24. Game \( G_{m0} \) is simply a rewrite of \( G_{ms}^{ms-uf} \), where \( H_0, H_1, H_2 \) are lazily sampled. We fix the given adversary \( A_{ms} \) for the rest of the proof and omit writing it in expression such as \( \Pr[G_{m0}(A_{ms})] \) for simplicity. Game \( G_{m1, \rho} \) is parameterized by a real number \( \rho \in [0, 1] \), and changes the code of \( NS, SIGN_1 \) and \( H_0 \). The changes are made so that \( SIGN_1 \) does not use the secret key \( sk \), but will however preserve the output distribution of all oracles when it does not abort, as we will show below. In particular, for each \( H_0 \) query, game \( G_{m1} \) makes a guess, by flipping a biased coin \( \text{Coin}(\rho) \), which has probability \( \rho \) of returning 1 and probability \( 1 - \rho \) of returning 0. If the coin flip returns 1, then we set the output of \( H_0(x) \) to be \( g^{\beta_g} pk^{\beta_{pk}} \), otherwise we set the output of \( H_0(x) \) to be \( g^{\beta_s} \). In either case, \( \beta_g \) and \( \beta_{pk} \) are uniformly chosen at random as per line 25.

Looking ahead, \( G_{m1, \rho} \) will be able to simulate signatures for \( pk, m \) when \( H_0(pk, m) \) is set to \( g^{\beta_g} pk^{\beta_{pk}} \) (when the coin toss returns 1). In fact, \( \rho \) is set to 1 in deriving the AGM result.
and the coin toss never returns 0. However, for the standard model result, we will need to make sure that the $H_0$ query corresponding to the forgery $pk, m$ is programmed differently, namely that $H_0((pk, m)) = g^{\beta_k}$.

Game $G_{1, \rho}$ could abort at line 16 (it is assumed that the adversary loses the game if
Gm₁ is aborted). By construction, we have

$$\Pr[\text{Gm}_1 \text{ does not abort}] = \rho^{g_0}.$$  \hspace{1cm} (3.57)

We claim that, for any value of $\rho$, if game Gm₁ does not abort, then it is indistinguishable from Gm₀ to the adversary. In particular, we claim

$$\Pr[\text{Gm}_1 \mid \text{Gm}_1 \text{ does not abort}] = \Pr[\text{Gm}_0].$$ \hspace{1cm} (3.58)

Showing this amounts to showing that the outputs of SIGN₁ oracle in either games are distributed identically. Observe that, in game Gm₀, the return value $T_v$ of NS and $(s_v, z_v)$ of SIGN₁ are uniformly distributed subjected to the constraint that

$$g^{\beta_v} H_0((pk_v, m))^{s_v} = T_{v,k} \cdot pk^{e_v}.$$ 

We will show that this is also true in Gm₁, ρ, namely that SIGN₀ and SIGN₁ in Gm₁,ρ also returns $T_{v,k}$ and $(s_v, z_v)$ that are uniformly distributed subjected to the above equation. In game Gm₁,ρ, if $w = pk$ at line 15, then $h = H_0((pk_v, m)) = g^{\beta_v} p_k^{\beta_k}$, by construction of $H_0$ (line 27). Hence, for a query SIGN₁$(v, (T_{v,1}, \ldots, T_{v,n}))$ of game Gm₁,ρ, it holds that

$$T_{v,k} \cdot pk^{e_v} = g^{a_v} \cdot h^{b_v} \cdot pk^{e_v} = g^{a_v} \cdot (g^{\beta_v} p_k^{\beta_k})^{b_v} \cdot pk^{e_v} = g^{a_v + \beta_v b_v} \cdot p_k^{\beta_k b_v + e_v}.$$ 

We claim that the above is also equal to $g^{\beta_v} \cdot h^{b_v}$. In fact, we set $z_v, s_v$ on line 17 and 18 exactly to
Additionally, notice that.

**Adversary $\mathcal{A}_{dl}(X)$:**

1. $vk \leftarrow X; (k, pk, m, (T, s, z)) \leftarrow \mathcal{A}_{ns,\text{Sign}_1,\text{Sign}_2, H_0, H_1, H_2}^N(vk)$
2. If $(pk[k] \neq X)$ then return ⊥
3. If $(pk, m) \in \{ (pk_i, m_i) : 1 \leq i \leq u \}$ then return ⊥
4. If not MS, $\forall f_{H_0, H_1, H_2}(pk, m, \sigma)$ then return ⊥
5. $(w, \beta_s, \beta_{pk}) \leftarrow TH(pk, m); apk \leftarrow \prod_{i=1}^{pk} pk[i]^j((i, pk))$
6. $c \leftarrow H_1((T, apk, m))$; For $i = 1, \ldots, |pk|$ do $e_i \leftarrow H_2((i, pk))$
7. $\alpha_g \leftarrow z + \beta_g - \text{Ext}(T, g) - c \cdot \sum_{i \neq k} \text{Ext}(pk[i], g) \cdot e_i$
8. $\alpha_X \leftarrow -s \cdot \beta_{pk} + \text{Ext}(T, X) + c \cdot (e_k + \sum_{i \neq k} \text{Ext}(pk[i], X) \cdot e_i)$
9. If $(\alpha_X = 0)$ then bad $\leftarrow$ true; $x' \leftarrow Z_p$
10. Else $x' \leftarrow \alpha_g \alpha_X^{-1}$ mod $p$
11. Return $x'$

**Figure 3.25:** Adversary $\mathcal{A}_{dl}$ for Theorem 3.7.1 oracles $\text{NS, Sign}_1, \text{Sign}_2, H_0, H_1, H_2$ are implemented using the exact code as those in $\text{Gm}_1,1$. Notation $\text{Ext}(. , g)$ and $\text{Ext}(., X)$ are defined in the proof of Lemma 3.7.2. Computation of $\alpha_g$ and $\alpha_X$ are done modulo $p$.

make this true. To verify this, check that

$$g^{s_v} h^v = g^{a_v + \beta_s b_v - \beta_s x_v} (g^{\beta_s p_k \beta_{pk}})^{s_v} = g^{a_v + \beta_s b_v + p_k \beta_{pk} x_v} = g^{a_v + \beta_s b_v \cdot p_k \beta_{pk} b_v + e_v c_v}.$$

Additionally, notice that $s_v, z_v$ are both marginally uniform over $Z_p$ by construction. This means the outputs of $\text{Sign}_0, \text{Sign}_1$ oracle from $\text{Gm}_1,1$ has the same output distribution compared to that of $\text{Gm}_0$. This justifies Equation (3.58).

Equipped with Lemma 3.16.1 we move on to prove Theorem 3.7.1. The proof constructs adversary $\mathcal{A}_{dl}$ that simulates $\text{Gm}_1,1$ (with $\rho$ set to 1).

**Proof of Theorem 3.7.1:** Consider the games $\text{Gm}_0$ and $\text{Gm}_1,1$ (with $\rho = 1$) in Fig. 3.24. We know that,

$$\Pr[\text{Gm}_0] = \Pr[\text{Gm}_1,1 \mid \text{Gm}_1,1 \text{ does not abort}].$$
Moreover,
\[ \Pr[\text{Gm}_{1,\rho} \text{ does not abort}] = \rho^{0} = 1, \]
when \( \rho = 1 \). Hence, game Gm\(_{1,1} \) never aborts and \( \Pr[\text{Gm}_0] = \Pr[\text{Gm}_{1,1}] \). We shall construct an adversary \( \mathcal{A}_{dl} \), using the fact that given adversary \( \mathcal{A}_{ms}^{\text{alg}} \) is algebraic, directly against game Gm\(_{G,g}^d\).

We first analyze the group elements involved in the inputs and outputs of oracles of Gm\(_{1,1} \). The \( u \)-th NS query takes in a list of group elements \( \text{pk}_u \). The \( v \)-th Sign\(_1 \) query takes in a list of group elements \( (T_{1,1}, \ldots, T_{v,n}) \). The \( i \)-th H\(_2 \) query take in a list of group elements \( \text{pk}_{H_2,i} \). The \( i \)-th H\(_1 \) query \( (T, \text{apk}, m) \) takes in group elements \( T_{H_1,i} \) and \( \text{apk}_{H_1,i} \). Above are the exhaustive list of group elements that are given to Gm\(_{1,1} \), let us denote this list by \( \text{out} \), since they are the output of the adversary. The initial query to \( \text{Init} \) outputs a group element \( \text{pk} \). The \( u \)-th NS query gives out a group element \( T_{u,k_u} \). The \( i \)-th H\(_0 \) query gives out a group element \( h_i \). The last query to \( \text{Fin} \) gives group elements \( T \) (first component of the forged signature) and \( \text{pk} \). Above (plus the group generator \( g \)) are the exhaustive list of group elements that are given out to the adversary \( \mathcal{A}_{ms}^{\text{alg}} \). Let us denote this list as \( \text{in} \). Hence, the algebraic adversary \( \mathcal{A}_{ms}^{\text{alg}} \) gives, for each group element in the list \( \text{out} \), a vector that is of dimension \( | \text{in} | \) which is a valid representation of the corresponding group element. Note that every group element in the list \( \text{in} \) is derived using only group operations on two group elements: \( g \) and \( \text{pk} \) (this is by the construction of game Gm\(_{1,1} \)). As a result, every group element in the list \( \text{out} \) can be represent using \( g \) and \( \text{pk} \) only. For any \( Y \in \text{out} \), we use Ext\((Y, g)\) and Ext\((Y, \text{pk})\) to denote this representation, i.e.

\[ Y = g^{\text{Ext}(Y, g)} \cdot \text{pk}^{\text{Ext}(Y, \text{pk})}. \]

We forego writing explicit code deriving these representations, with the understanding that they are well-defined and can be computed easily from the oracle queries of \( \mathcal{A}_{ms}^{\text{alg}} \). We will use this notation freely in simulations of Gm\(_{1,1} \).

We move on to giving adversary \( \mathcal{A}_{dl} \), which simulates Gm\(_{1,1} \) for \( \mathcal{A}_{ms}^{\text{alg}} \). Our adversary \( \mathcal{A}_{dl} \) is
given in Fig. 3.25. Our adversary $A_{dl}$ simulates oracles $NS, SignStage_1, SignStage_2, H_0, H_1$ exactly as $Gm_{1,1}$, hence their code are omitted. As stated above, since $A_{dl}$ simulates $Gm_{1,1}$, the representation of any group element $Y \in out$ are available via scalars $Ext(Y, g)$ and $Ext(g, pk)$. Our adversary uses these scalars to compute the discrete log $x'$.

If $A_{ms}^{alg}$ gives a valid forgery $(pk, m, (T, s, z))$, then the verification equation says that

$$g^z H_0((pk, m)) = T \cdot apk^H_1((T, apk, m)),$$

where $apk = \prod_{i=1}^{p|k|} pk[i]^{H_2((i, pk))}$. Since every group element in the above equation can be represented using $g$ and $X$, one can solve for $DL_{(g, X)}(X)$. Our adversary $A_{dl}$ implements this intuition, computing value $\alpha_g$ and $\alpha_X$ (line 7 and 8) such that $g^{\alpha_g} = X^{\alpha_X}$. The only caveat is that $\alpha_X$ could be 0, in which case $DL_{(g, X)}(X)$ cannot be solved for. When $\alpha_X = 0$ adversary $A_{dl}$ sets bad, and we would like to upperbound the probability of this event. First, note that the view of adversary $A_{ms}$ is independent of the value of $\beta_{pk}$. This is because the adversary is only given the value of $h = g^{b_s} pk^{\beta_{pk}}$. So, if the forgery is such that $s \neq 0$, then $\alpha_X = 0$ with probability at most $1/p$. If $s = 0$, then we need to make sure that $Ext(T, X) + c \cdot (\sum_{i \neq k} Ext(pk[i], X) \cdot e_i)$ is not zero. We first bound the probability that there exists some query $H_2((\cdot, pk'))$ (which defines the values of $e'_{p|k'|}$) such that $e'_k + \sum_{i \neq k} Ext(pk'[i], X) \cdot e'_i = 0$ (call this quantity $\gamma_{pk'}$). This happens with probability at most $q_2/p$. Suppose the above does not happen, then for each query $H_1((T', apk', m'))$ (which defines the value of $c'$), where $apk'$ is the aggregate key of some vector $pk'$, the probability that $Ext(T', X) + c' \cdot \gamma_{pk'} = 0$ is at most $q_2/p$, accounting for at most $q_2$ non-zero values that $\gamma_{pk'}$ could take. This results in an overall bad probability of $q_2/p + q_1 q_2/p = (q_1 + 1) q_2/p$. This justifies Equation (3.7).
**Figure 3.26:** Games Gm\(_3\) and Gm\(_4\) for proof of Theorem 3.7.2. Oracles Init, NS, Sign\(_1\), Sign\(_2\), and H\(_0\) are the same as those in Gm\(_2\), \(\rho\). Parameter \(\rho\) is set to \((1 - (1 + q_s)^{-1})\) in oracle H\(_0\).

### 3.17 Proof of Theorem 3.7.2

**Proof of Theorem 3.7.2:** We will start by considering Gm\(_1\), \(\rho\) given in Fig. 3.24. By Lemma 3.16.1,

\[
\text{Adv}^{\text{ms-uf}}_{\text{MS}}(A_{\text{ms}}) = \Pr[\text{Gm}_{1, \rho}(A_{\text{ms}}) \mid \text{Gm}_{1, \rho}(A_{\text{ms}}) \text{ does not abort}].
\]

Towards construction of an adversary against XIDL, consider game Gm\(_2\), \(\rho\) (Fig. 3.24), differ from Gm\(_1\), \(\rho\) only at line 40—it aborts if the coin flip corresponding to the forgery target \((pk, m)\) results in \(w = g\). Marginally, Gm\(_2\), \(\rho\) does not abort at line 40 with probability \((1 - \rho)\). We need to lower bound the probability of Gm\(_2\), \(\rho\) not aborting overall, at either line 16 or line 40. Since there are overall \(q_s\) unique queries to NS in the execution of Gm\(_0\) with \(A_{\text{ms}}\), then the probability that Gm\(_1\)
**Figure 3.27**: Adversary $A_{\text{kidl}}$ used in Theorem 3.7.2. Oracles $NS, \text{SIGN}_1, \text{SIGN}_2, H_0$ are simulated exactly per code from Fig. 3.24.

does not abort is exactly

$$\text{Pr}[\text{Gm}_2(A_{\text{ms}}) \text{ does not abort}] = \rho^{qs} (1 - \rho) .$$

Setting $\rho = (1 - (1 + q_s)^{-1})$, we have that

$$\text{Pr}[\text{Gm}_2(A_{\text{ms}}) \text{ does not abort}] = (1 - (1 + q_s)^{-1})^{qs} (1 + q_s)^{-1} \geq \frac{1}{e(1 + q_s)} ,$$

180
where we applied the fact that \((1 - (1+n)^{-1})^n \geq e^{-1}\) for positive \(n\). Since game \(G_{m2}\) can only abort more often than \(G_{m1}\) and that the aborting at line 40 is an event independent of whether \(A_{ms}\) succeeds, Equation (3.58) gives us that

\[
\Pr[G_{m0}(A_{ms})] = \Pr[G_{m2}(A_{ms}) | G_{m2}(A_{ms}) \text{ does not abort}].
\]

Hence,

\[
\Pr[G_{m2, \rho}(A_{ms})] \geq \frac{1}{e(1+q_s)} \cdot \Pr[G_{m0}(A_{ms})]. \tag{3.59}
\]

For the rest of the proof, we set \(\rho = (1 - (1+q_s)^{-1})\) and omit writing them in the subscript for games. Next, we need to further modify oracles \(H_1\) and \(H_2\) so that whenever \(H_2\) derives a fresh aggregate key \(apk\), it must not have been queried to \(H_1\) (in the form of \((T, apk, m)\) for any \(T\) and \(m\)). Formally, consider games \(G_{m3}\) and \(G_{m4}\) given in Fig. 3.26 These games also keep track of a set \(BadSet\), which contains those \(H_1\) queries \((T, apk, m)\) such that the aggregate key \(apk\) is later derived in \(H_2\) (line 51). By construction, if any \(H_1\) query \((T, apk, m)\) is not in \(BadSet\) (at the end of the game execution), the aggregate key \(apk\) is either previosly derived in \(H_2\), or it has never been derived in any \(H_2\) query. Game \(G_{m3, Fin}\) does not contain the boxed code, which makes the oracle behave identically to \(G_{m2, H_2}\). So, we have

\[
\Pr[G_{m2}(A)] = \Pr[G_{m3}(A)]. \tag{3.60}
\]

Oracle \(G_{m4, H_2}\) contains the boxed code, which reset the oracle \(H_1\) at the chosen forgery point \((T, apk, m)\) if it is part of \(BadSet\). This ensures the value \(HF_1[(T, apk, m)]\) to always be defined after the \(H_2\) query that derives aggregate key \(apk\). By construction, \(G_{m3}\) and \(G_{m4}\) are identical-until-bad. So,

\[
\Pr[G_{m3}(A)] \leq \Pr[G_{m4}(A)] + \Pr[G_{m4} \text{ sets bad}]. \tag{3.61}
\]
We first compute that probability that $\text{BadSet}$ is non-empty at line 57. Since each $H_2$ query has probability at most $q_1/p$ probability of adding elements to $\text{BadSet}$, we can bound

$$\Pr[\text{BadSet} \neq \emptyset \text{ at line 57}] \leq \frac{q_1q_2}{p}.$$  \hspace{1cm} (3.62)

Note that flag bad can only be set if $G_{m_4}$ did not abort (in oracle $H_0$ or line 55), which happens with probability $1/(e(1+q_s))$ by previous analysis. Furthermore, the view of the adversary is independent of whether game $G_{m_4}$ aborts. Hence,

$$\Pr[G_{m_4}(A) \text{ sets bad}] \leq \frac{q_1q_2}{ep(1+q_s)}. \hspace{1cm} (3.63)$$

We now move on to the construction of the adversary, given in Fig. 3.27. The adversary $A_{xidl}$ runs $A_{ms}$ while giving it simulated oracle $H_0, H_1, H_2, NS, \text{SIGNSTAGE}_1, \text{SIGNSTAGE}_2$. Code for $H_0, NS, \text{SIGN}_1, \text{SIGN}_2$ are copied from game $G_{m_4}$. The only new code here is in $H_1$ and $H_2$, which we now explain.

For each $j$-th $H_2$ query $x = (\cdot, pk)$, where $HF_2[x]$ is not yet defined the adversary will sample $HF_2[(i, pk)]$ for each $i = 1, \ldots, |pk|$ as follows. If the target public key $X$ is not in $pk$, then these values are sampled honestly (line 15). Otherwise, let $k$ be the smallest index such that $pk[k] = X$. Our adversary will query the NWTar oracle from $G_{m_4}^{xidl}$ so that the resulting aggregate public key $apk$ is the target point $T_j$ generated by the game $G_{m_4}^{xidl}$. This is done by first computing the partial aggregation value of $S$ (line 17), before submitting it to the NWTar oracle to obtain response $e_j$ which is set as the output of $H_2$ (line 19).

For each $H_1$ query $(T, apk, m)$, the adversary will submit the commitment to the oracle $Ch$, at the index that corresponds to the aggregate public key $apk$. This is done so that a forgery $(T, s, z)$ corresponding to this $H_1$ query can be turned into a break against $G_{m_4}^{xidl}_{C,g,q_2,q_1}$. Here, we are also utilizing the fact that a successful forgery $(pk, m, (T, s, z))$ is such that $H_0((pk, m))$ is a
known power of \( g \). Hence, the verification equation

\[ g^zh^s = T \cdot apkH_1((T,apk,m)), \]

of the signature scheme implies that the computed response \( z + \beta g^s \), against the game \( Gm_{\text{xidl},g,q_2,q_1} \), is valid, i.e. \( g^zh^s = T \cdot T_jH_1((T,apk,m)) \), where \( T_j = apk \) is the \( j \)-th target point generated by \( \text{NwTAR} \) oracle. Hence,

\[
\Pr[Gm_4(\mathcal{A}_{ms})] = \Pr[Gm_{\text{xidl},g,q_2,q_1}(\mathcal{A}_{\text{xidl}})]. \tag{3.64}
\]

Putting Equation (3.59), (3.60), (3.61) and (3.64) together, we obtain the result claimed in the theorem. \( \square \)

### 3.18 Acknowledgements

We thank the ASIACRYPT 2021 reviewers for their careful reading and valuable comments.

Bellare was supported in part by NSF grant CNS-1717640 and a gift from Microsoft. Dai was supported in part by a Powell Fellowship and grants of the Bellare.

Chapter 3 in full, is a reprint of the material as it appears in International Conference on the Theory and Application of Cryptology and Information Security, Asiacrypt 2021. Bellare, Mihir; Dai, Wei. Springer, 2021. The dissertation author was the primary investigator and author of this paper.
Bibliography


[9] James Bartusek, Fermi Ma, and Mark Zhandry. The distinction between fixed and random

non-rewinding proofs for Schnorr identification and signatures. In Karthikeyan Bhargavan,
Elisabeth Oswald, and Manoj Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of


[12] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signa-

one-more-RSA-inversion problems and the security of Chaum’s blind signature scheme.

[14] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and
a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di

[15] Mihir Bellare and Adriana Palacio. GQ and Schnorr identification schemes: Proofs of
security against impersonation under active and concurrent attacks. In Moti Yung, editor,

[16] Mihir Bellare and Adriana Palacio. The knowledge-of-exponent assumptions and 3-round
zero-knowledge protocols. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of

[17] Mihir Bellare, Bertram Poettering, and Douglas Stebila. From identification to signatures,
tightly: A framework and generic transforms. In Jung Hee Cheon and Tsuyoshi Takagi,
Heidelberg, December 2016.

[18] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing
efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu,

[19] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for


