Ring Trapdoor Functions: A Lattice-Based Framework for Secure Ring Signatures

Capstone Thesis by
Bhumika Mittal

In Partial Fulfillment of the Requirements

For the Postgraduate Diploma in

Advanced Studies and Research in Computer Science

Department of Computer Science
Ashoka University, India

Thesis Supervisors:
Eike Kiltz, Professor of Computer Science, Ruhr-Universität Bochum
Mahavir Jhawar, Associate Professor of Computer Science, Ashoka University

CERTIFICATE

This is to certify that the capstone thesis report titled, **"Ring Trapdoor Functions: A Lattice-Based Framework for Secure Ring Signatures"**, submitted by **Bhumika Mittal**, to the Department of Computer Science, Ashoka University, for the award of the Postgraduate Diploma in **Advanced Studies and Research in Computer Science**, is a bona fide record of the research work carried out by her under our supervision. The contents of this thesis, in full or in part, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Eike Kiltz
Professor of Computer Science
Ruhr-Universität Bochum

Dr. Mahavir Jhawar
Associate Professor of Computer Science
Ashoka University

Place: Ashoka University, India Date: **30th April 2025**

Contents

Preface

Motivation

A central line of work in theoretical cryptography is to provide formal (and strong) definitions of security for cryptographic primitives, and then provide constructions, based on general computational complexity assumptions (such as the existence of one-way or trapdoor functions), that satisfy the definitions in question. The value and difficulty of such "foundational" work is acknowledged and manifest.

This work presents a generic framework for constructing efficient ring signature schemes through novel cryptographic abstractions. Introduced by Rivest, Shamir, and Tauman [RST01] as an extension of group signatures [CVH91], ring signatures enable users to sign messages on behalf of dynamically formed groups while preserving signer anonymity within the ring ρ. This primitive has found widespread adoption in blockchain systems (Monero, Bytecoin), electronic voting, and privacy-preserving protocols.

While numerous constructions exist under number-theoretic assumptions [BSW02,Nao02, AOS02,**?**,BGLS03], recent research focuses on post-quantum alternatives. Lattice-based schemes [BK10, AMBB$^+$13, LLNW16, BLO18, ESS$^+$19, BKP20, LNS21] dominate this space, with proof system-based approaches [ESS$^+$19, BKP20, LNS21] achieving asymptotic signature sizes of $\mathcal{O}(\log |\rho|)$. However, these come with substantial overhead—concrete implementations range from 16 KB ($|\rho| \leq 32$) to 22 KB ($|\rho| = 2^{25}$). Notable existing schemes include `GANDALF` (optimized for small rings) [GJK24], `RAPTOR` [LAZ19] (2.5 KB for $|\rho| = 2$), and `DualRing` [YEL$^+$21] (optimal for $4 \leq |\rho| \leq 439$).

We introduce an additional abstraction layer through *Ring Trapdoor Functions* (RTDFs) for three key reasons:

1. **Foundational Insight**: Enables fine-grained analysis of ring signature components and their security dependencies
2. **Modular Improvement**: Permits black-box enhancements (e.g., ring size scaling) without protocol redesign
3. **Primitive Reusability**: Provides building blocks for other anonymity-preserving cryptosystems

While the paper [BK10] first proposed ring trapdoor functions, our RTDF formulation offers greater flexibility. The BK10 definition of ring trapdoor functions, while foundational, imposes structural constraints that limit its generality and fail to capture constructions like GANDALF.

Acknowledgements

I would like to express my gratitude to my thesis supervisors, Eike Kiltz and Mahavir Jhawar, for their guidance, support, and encouragement throughout the course of this work. I am especially grateful to Jonas Janneck, a third-year PhD student at Ruhr-Universität Bochum, for his invaluable mentorship, insightful feedback, and suggestions, which greatly contributed to the development of this thesis. Finally, I would like to thank the members of the PAC committee for their time and effort in reviewing my work.

1

Introduction

1.1 Notations

We write $s \xleftarrow{\$} S$ to denote the uniform sampling of s from the finite set S. For an integer n, we define $[n] := \{1, \ldots, n\}$. The notation $[\![b]\!]$, where b is a boolean statement, evaluates to 1 if the statement is true and 0 otherwise. We use uppercase letters $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}$ to denote algorithms.

Unless otherwise stated, algorithms are probabilistic, and we write $(y_1, \ldots) \xleftarrow{\$} \mathcal{A}(x_1, \ldots)$ to denote that $\mathcal{A}$ returns $(y_1, \ldots)$ when run on input $(x_1, \ldots)$. We write $\mathcal{A}^{\mathcal{B}}$ to denote that $\mathcal{A}$ has oracle access to $\mathcal{B}$ during its execution. For a randomized algorithm $\mathcal{A}$, we use the notation $y \xleftarrow{\$} \mathcal{A}(x)$ to denote that y is a possible output of $\mathcal{A}$ on input x. The support of a discrete random variable X is defined as $\mathrm{supp}(X) := \{x \in \mathbb{R} \mid \Pr[X = x] > 0\}$. For two polynomials f, g, we denote the polynomial multiplication of f and g by $f * g$.

1.2 Preliminaries

Functions. We use functions to describe mappings between sets. A function $f : X \to Y$ maps each element $x \in X$ to a unique element $y \in Y$. The set X is called the *domain* of f, and the set Y is called the *codomain*. The value of f at x is denoted by $f(x)$.

A function is said to be:

- **Injective** (one-to-one) if for all $x_1, x_2 \in X$, $f(x_1) = f(x_2)$ implies $x_1 = x_2$.
- **Surjective** (onto) if for every $y \in Y$, there exists $x \in X$ such that $f(x) = y$.
- **Bijective** if it is both injective and surjective, i.e., a one-to-one correspondence between X and Y.

A function f is called **efficiently computable** if there exists a deterministic polynomial-time algorithm that, given $x \in X$, outputs $f(x)$. We often consider function families $\{f_k\}$ parameterized by a key k. If $f_k : X \to Y$, then f_k denotes the function obtained by fixing key k. In cryptographic contexts, such keys are often sampled uniformly at random from some key space $\mathcal{K}$.

Probabilities. We collect a number of basic definitions and simple facts about probabilities.

Let A and B be events in some discrete probability space.

- **Complementary event.** $\neg A = \overline{A}$ denotes the event that A does not happen, i.e., $\Pr[\neg A] = 1 - \Pr[A]$.
- **Conditional probability.** If $\Pr[B] > 0$, then

$$\Pr[A \mid B] := \frac{\Pr[A \wedge B]}{\Pr[B]}$$

  denotes the probability that A happens conditioned on the event that B already happened.

- **Independence.** A and B are independent events if

$$\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B].$$

- **Union Bound (Boole's inequality).**

$$\Pr[A \vee B] \leq \Pr[A] + \Pr[B].$$

  More generally,

$$\Pr\left[\bigvee A_i\right] \leq \sum \Pr[A_i].$$

- **Total probability.**

$$\Pr[A] = \Pr[A \wedge B] + \Pr[A \wedge \neg B] = \Pr[A \mid B] \cdot \Pr[B] + \Pr[A \mid \neg B] \cdot \Pr[\neg B].$$

Algorithms. If A is a set, then $a \xleftarrow{\$} A$ denotes picking a from set A according to the uniform distribution.

An algorithm $\mathcal{A}$ is a probabilistic Turing machine which is usually specified using pseudo-code. We say that $\mathcal{A}$ is a probabilistic polynomial-time (PPT) algorithm if its running time is bounded by a fixed polynomial in its input size. If b is a string, then $a \xleftarrow{\$} \mathcal{A}(b)$ denotes the output of algorithm $\mathcal{A}$ when run on input b. Note that a is a random variable whose distribution is induced by algorithm $\mathcal{A}$. (If $\mathcal{A}$ is deterministic, then a is a fixed value and we write $a := \mathcal{A}(b)$.) The expression $\mathcal{A}(a) \Rightarrow b$ denotes the event that algorithm $\mathcal{A}$ outputs b on input a. We are usually interested in $\Pr[\mathcal{A} \Rightarrow 1]$, the probability that $\mathcal{A}(a)$ outputs 1.

Adversaries $\mathcal{A}$, oracles $\mathcal{O}$, and procedures $\mathcal{P}$ are used as synonyms for algorithms. We use the different terms only to refer to different contexts.

Let $\mathcal{A}$ be an adversary and let $\pi(\cdot)$ be an oracle. Then $A^{\pi(\cdot)}$ denotes that adversary $\mathcal{A}$ has oracle access to $\pi(\cdot)$, i.e., $\mathcal{A}$ can make an arbitrary number of queries x_i to its oracle that are answered with $\pi(x_i)$.

Security Games. We use standard code-based security games [BR04]. A *game* $\mathcal{G}$ is a probability experiment in which an adversary $\mathcal{A}$ interacts with an implicit challenger that answers oracle queries issued by $\mathcal{A}$. The game $\mathcal{G}$ has one *main procedure* and an arbitrary amount of additional *oracle procedures* which describe how these oracle queries are answered.

We denote the (binary) output b of game $\mathcal{G}$ between a challenger and an adversary $\mathcal{A}$ as $\mathcal{G}(\mathcal{A}) \Rightarrow b$. $\mathcal{A}$ is said to *win* $\mathcal{G}$ if $\mathcal{G}(\mathcal{A}) \Rightarrow 1$, or shortly $\mathcal{G} \Rightarrow 1$. Unless otherwise stated, the randomness in the probability term $\Pr[\mathcal{G}(\mathcal{A}) \Rightarrow 1]$ is over all the random coins in game $\mathcal{G}$.

If a game is aborted, the output is either 0 or a random bit b in case of an indistinguishability game, i.e., a game for which the advantage of an adversary is defined as the absolute difference of winning the game to $\frac{1}{2}$. To provide a cleaner description and avoid repetitions, we sometimes refer to procedures of different games. To call the oracle procedure `Oracle` of game $\mathcal{G}$ on input x, we shortly write $\mathcal{G}.\texttt{Oracle}(x)$.

Lattice-based cryptography. Lattice-based cryptography relies on the computational hardness of certain problems in high-dimensional lattices, such as the Shortest Vector Problem (SVP) and the Learning With Errors (LWE) problem. These foundations enable post-quantum secure cryptographic constructions.

Definition 1.2.1 (Lattice)**.** For any set of linearly independent vectors $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\} \in \mathbb{R}^{m \times n}$, the lattice generated by $\mathbf{B}$ is defined as:

$$\mathcal{L} = \mathcal{L}(\mathbf{B}) := \left\{ \sum_{i=1}^{n} z[i]\mathbf{b}_i \mid z[i] \in \mathbb{Z} \right\}.$$

Here, n is called the rank of the lattice and m is the dimension. A lattice can have infinitely many bases. $\mathbf{B}$ is called a basis of $\mathcal{L}$ if $\mathcal{L} = \mathcal{L}(\mathbf{B})$.

Definition 1.2.2 (NTRU Lattice)**.** Let $N = 2^k$ for $k \in \mathbb{Z}$, q prime, $f, g \in R = \mathbb{Z}[X]/(X^N + 1)$, and $h = g * f^{-1} \mod q$. The NTRU lattice parameterized by h and q is a lattice of volume q^N in $\mathbb{R}^{2N}$ in the coefficient embedding of the following module

$$\Lambda_{h,q} := \{(u, v) \in \mathbb{R}^2 : u * h + v = 0 \mod q\}.$$

Equivalently, for $R = \mathbb{Z}[X]/(X^N + 1)$, an NTRU lattice is a full-rank submodule lattice of $\mathbb{R}^2$ generated by the columns of a matrix of the form

$$B_h = \begin{bmatrix} 1 & 0 \\ h & q \end{bmatrix}$$

for prime q and some $h \in R$ coprime to q. A trapdoor for this lattice is a relatively short basis

$$B_{f,g} = \begin{bmatrix} f & F \\ g & G \end{bmatrix}$$

where the basis vectors $(f, g) \in \mathbb{R}^2$ and $(F, G) \in \mathbb{R}^2$ are not much larger than $\sqrt{\det B_h} = \sqrt{q}$ and $f * G - g * F = q$.

Norms. For a polynomial $f \in R_q = \mathbb{Z}_q[X]/(X^N+1)$, let $f \in \mathbb{Z}_q^N$ denote the coefficient embedding of f, and $f_i \in \mathbb{Z}_q$ the i-th coefficient. For an element $f_i \in \mathbb{Z}_q$, we write $|f_i|$ to denote $|f_i \mod q|$. Let the ℓ_2-norm for $f = f_0 + f_1 X + \ldots + f_{N-1}X^{N-1} \in R_q$ be defined as $\|f\|_2 := \sqrt{\sum_{i=0}^{N-1} |f_i|^2}$.

For polynomials $f_1, \ldots, f_k \in R_q$, we use the notation

$$\|(f_1, \ldots, f_k)\|_2 := \sqrt{\sum_{i=0}^{N-1} (|f_{1,i}|^2 + \ldots + |f_{k,i}|^2)}.$$

Discrete Gaussian Distribution over Λ. For any standard deviation $s > 0$, the n-dimensional *Gaussian function* $\rho_{s,c} : \mathbb{R}^n \to (0, 1]$ on $\mathbb{R}^n$ centred at $c \in \mathbb{R}^n$ with standard deviation s is defined by

$$\rho_{s,c}(x) := \exp\left(\frac{-\|x-c\|_2^2}{2s^2}\right).$$

For any $c \in \mathbb{R}^n$, $s \in \mathbb{R}^+$, and lattice Λ, the *discrete Gaussian distribution over* Λ is defined as

$$\forall x \in \Lambda, \quad D_{\Lambda,s,c} := \frac{\rho_{s,c}(x)}{\sum_{z\in\Lambda} \rho_{s,c}(z)}.$$

We omit the subscript c when the Gaussian is centred at 0 and subscript Λ when the Gaussian is over $\mathbb{Z}^n$.

Ring Learning With Errors (RLWE). Let $R := \mathbb{Z}[X]/(X^N+1)$. The *Ring Learning With Errors* problem relative to the NTRU trapdoor algorithm `TpdGen` with parameters $m, q, \alpha > 0$ and $s \geq 0$ is defined via the game R-LWE, depicted below. We define the advantage of $\mathcal{A}$ in R-LWE as

$$\mathrm{Adv}^{R\text{-LWE}}_{m,q,\alpha,s,\mathcal{A}} := \left|\Pr\left[R\text{-LWE}_{m,q,\alpha,s}(\mathcal{A}) \Rightarrow 1\right] - \frac{1}{2}\right|.$$

Game R-LWE$_{m,q,\alpha,s}(\mathcal{A})$

$b \xleftarrow{\$} \{0,1\}$
$u \xleftarrow{\$} D_{\mathbb{Z}_N,s}$
for $i \in [m]$ **do**
  $(h_i, \cdot) \leftarrow \mathtt{TpdGen}(q, \alpha)$
  $v \xleftarrow{\$} D_{\mathbb{Z}_N,s}$
  if $b = 0$ **then**
    $z_i := u * h_i + v$
  else
    $z_i \xleftarrow{\$} R_q$
  end if
end for

12: $b' \xleftarrow{\$} \mathcal{A}((h_1, z_1), \ldots, (h_m, z_m))$
13: **return** $[b = b']$

Inhomogeneous Ring Short Integer Solution Problem (R-ISIS). Let $R := \mathbb{Z}[X]/(X^N + 1)$. The *Inhomogeneous Ring Short Integer Solution* problem relative to the NTRU trapdoor algorithm `TpdGen` with parameters $m, q > 0$ and $\alpha, \beta > 0$ is defined via the game R-ISIS, depicted below. We define the advantage of $\mathcal{A}$ in R-ISIS as

$$\mathrm{Adv}^{R\text{-ISIS}}_{m,q,\alpha,\beta,\mathcal{A}} := \Pr\left[R\text{-ISIS}_{m,q,\alpha,\beta}(\mathcal{A}) \Rightarrow 1\right].$$

Game R-ISIS$_{m,q,\alpha,\beta}(\mathcal{A})$
1: **for** $i \in [m]$ **do**
2: $\quad (h_i, \cdot) \leftarrow \texttt{TpdGen}(q, \alpha)$
3: **end for**
4: $c \xleftarrow{\$} R_q$
5: $(u_1, \ldots, u_m, v) \xleftarrow{\$} \mathcal{A}(h_1, \ldots, h_m, c)$
6: **return** $\left[\sum_{i\in[m]} h_i * u_i + v = c \wedge \|(u_1, \ldots, u_m, v)\|_2 \leq \beta\right]$

2

One-Way Functions and Signature Schemes

One-way functions (OWFs) are a fundamental building block in cryptography, serving as the basis for various cryptographic primitives, including digital signatures. In this chapter, we will explore the definition of one-way functions, their properties, and their role in constructing secure signature schemes. We will also discuss Rompel's foundational work that established the necessity and sufficiency of one-way functions for secure digital signatures.

2.1 Definition of One-Way Functions

2.1.1 Syntax

A *one-way function* (OWF) is defined by two algorithms $(\mathsf{Gen}, \mathsf{Eval})$:

$a \xleftarrow{\$} \mathsf{Gen}(1^\kappa)$: Given a security parameter κ, the probabilistic generation algorithm outputs a function description a, defining a function $f_a : \mathcal{D} \to \mathcal{R}$.

$y \leftarrow \mathsf{Eval}(a, x)$: The deterministic evaluation algorithm computes $y = f_a(x)$ for $x \in \mathcal{D}$.

2.1.2 Properties

1. **Correctness:** An OWF is $\delta(\kappa)$-correct if for all $\kappa \in \mathbb{N}$ and $a \xleftarrow{\$} \mathsf{Gen}(1^\kappa)$:

$$\Pr\left[\mathsf{Eval}(a, x) \neq f_a(x)\right] \leq \delta(\kappa).$$

2. **One-Wayness:** For any PPT adversary $\mathcal{A}$, the advantage function of $\mathcal{A}$ in breaking the one-wayness of f_a is defined as:

$$\mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{A}}(\kappa) := \Pr\left[\mathsf{OWF}^{\mathcal{A}}(\kappa) = 1\right],$$

where the security experiment $\mathsf{OWF}^{\mathcal{A}}(\kappa)$ is:

Game $\mathsf{OWF}^{\mathcal{A}}(\kappa)$

1: $a \xleftarrow{\$} \mathsf{Gen}(1^{\kappa})$
2: $x \xleftarrow{\$} \mathcal{D}$
3: $y \leftarrow \mathsf{Eval}(a, x)$
4: $x' \xleftarrow{\$} \mathcal{A}(a, y)$
5: **return** 1 if $\mathsf{Eval}(a, x') = y$, else 0

2.2 Definition of Signature Scheme

Signature schemes are cryptographic primitives that allow a user to sign messages, providing authenticity and integrity. A signature scheme consists of three algorithms: key generation, signing, and verification.

2.2.1 Syntax

A signature scheme is a tuple $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ of ppt algorithms such that:

- $\mathsf{Gen}(1^{\kappa})$: On input security parameter κ, the key generation algorithm outputs a public key pk and a secret key sk.
- $\mathsf{Sign}(sk, \mu)$: Given a secret key sk and a message $\mu \in \mathcal{M}$ (where $\mathcal{M}$ is the message space), outputs a signature $\sigma \in \{0,1\}^*$.
- $\mathsf{Ver}(pk, \mu, \sigma)$: Given pk, message μ, and signature σ, outputs 1 (accept) or 0 (reject).

2.2.2 Security

1. **Correctness:** For all $\kappa \in \mathbb{N}$, $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^{\kappa})$, and $\mu \in \mathcal{M}$, the signature scheme satisfies:
$$\Pr\left[\mathsf{Ver}(pk, \mu, \mathsf{Sign}(sk, \mu)) \neq 1\right] \leq \delta(\kappa)$$
where the probability is over Gen and Sign's randomness.

2. **Existential Unforgeability (EUF-CMA):** For any ppt adversary $\mathcal{A}$, the advantage is defined as:
$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{S},\mathcal{A}}(\kappa) := \Pr\left[\mathsf{EUF\text{-}CMA}^{\mathcal{A}}(\kappa) = 1\right]$$
where the security experiment $\mathsf{EUF\text{-}CMA}^{\mathcal{A}}(\kappa)$ is defined as follows:

Game $\mathsf{EUF\text{-}CMA}^{\mathcal{A}}(\kappa)$

1: $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^\kappa)$
2: $\mathcal{Q} \leftarrow \emptyset$
3: $(\mu^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{SignO}(\cdot)}(pk)$
4: **return** $[[\mathsf{Ver}(pk, \mu^*, \sigma^*) = 1 \wedge \mu^* \notin \mathcal{Q}]]$

Oracle $\mathsf{SignO}(\mu)$

1: $\sigma \xleftarrow{\$} \mathsf{Sign}(sk, \mu)$
2: $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\mu\}$
3: **return** σ

The probability is taken over $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^\kappa)$, $\mathcal{A}$'s randomness, and SignO's responses.

2.3 Rompel's Signature Framework

The foundational work of Rompel [Rom90] established that the existence of *one-way functions* (OWFs) is both necessary and sufficient for constructing signature schemes secure against *existential forgery under adaptive chosen-message attacks* (EUF-CMA). This result resolves a long-standing open problem in theoretical cryptography, showing that OWFs—a minimal cryptographic primitive—are the sole requirement for achieving the strongest notion of signature security.

Rompel's framework proceeds in two key stages:

1. **Necessity**: Any EUF-CMA-secure signature scheme implies the existence of OWFs.
2. **Sufficiency**: A generic construction of signatures from *any* OWF, via a novel transformation to *one-way hash functions*.

The construction leverages a series of carefully designed function families to amplify weak security properties into full-fledged EUF-CMA security. Below, we formalize these results and their proof techniques.

Theorem 2.3.1. *The existence of a signature scheme secure against existential forgery under adaptive chosen message attacks implies the existence of one-way functions.*

Proof. Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ be a secure signature scheme. Define function f as:

$$f(1^\kappa) = pk \quad \text{where} \quad (pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$$

Assume f is not one-way. Then there exists a PPT adversary $\mathcal{A}$ that inverts f with non-negligible probability. This adversary could:

1. Given pk, compute $sk' \leftarrow \mathcal{A}(pk)$ such that (pk, sk') is a valid key pair
2. Forge signatures using sk', violating $\mathcal{S}$'s security

Thus, f must be one-way. □

The paper also provides a construction of a signature scheme from any one-way function. The construction is based on the idea of using the one-way function to create a hash function, which is then used to sign messages. The key idea is to use the one-way function to create a pseudorandom generator that can be used to generate signatures.

2.4 Conclusion

While Rompel's result remains foundational in establishing the minimal cryptographic assumptions required for digital signatures, its practical use is limited. As a result, real-world systems rely on *trapdoor functions* (TDFs).

This dichotomy arises from the efficiency gaps between OWF-based and TDF-based signature schemes. OWF-based signatures (e.g., hash-based schemes) produce signatures that grow linearly with the security parameter. In contrast, TDF-based schemes like RSA and ECDSA yield constant-size signatures. OWF-based schemes require polynomial-time verification (e.g., Merkle tree traversals), while TDFs enable constant-time checks via algebraic operations. OWF constructions need heavy precomputation during key generation, unlike TDFs, which support efficient trapdoor sampling.

The following chapter will formally define trapdoor functions and analyze their role in developing efficient, deployable signature schemes.

3

Trapdoor Functions

This chapter introduces the concept of trapdoor functions (TDFs) and their application in constructing signature schemes. TDFs are a fundamental building block in cryptography, enabling secure communication and authentication. The key idea behind TDFs is that they are easy to compute in one direction (using the public key) but hard to invert without a special piece of information (the trapdoor). This property is crucial for creating secure digital signatures, which allow users to sign messages in a way that can be verified by others without revealing the signing key.

3.1 Definition of Trapdoor Functions

3.1.1 Syntax

A *trapdoor function (TDF)* scheme consists of the following polynomial-time algorithms:

- $\mathsf{Gen}() \to (a, t)$: Generates a public function index a and trapdoor t.
- $\mathsf{Eval}(a, x) \to y$: Evaluates the function $f_a : \mathcal{D}_a \to \mathcal{R}_a$ on input x, outputting y.
- $\mathsf{Inv}(t, y) \to x$: Inverts y using trapdoor t to recover x.

3.1.2 Properties

1. **Correctness:** A TDF scheme is said to be *correct* the inversion algorithm always recovers the original input when given the trapdoor. Formally:

$$\Pr\begin{bmatrix}(a,t) \leftarrow \mathsf{Gen}(),\\ y \leftarrow \mathsf{Eval}(a,x),\\ x' \leftarrow \mathsf{Inv}(t,y) :\\ x' = x\end{bmatrix} = 1.$$

   This must hold for all $x \in \mathcal{D}_a$, where $\mathcal{D}_a$ is the domain of function f_a.

2. **One-Wayness:** The scheme is *one-way* if no PPT adversary can invert a randomly evaluated point without the trapdoor. This is formalized through the following security game $\mathsf{TDF\text{-}OWA}$

Game $\mathsf{TDF\text{-}OWA}_{\mathcal{A}}()$**:**
1: $(a, t) \leftarrow \mathsf{Gen}()$
2: $x \xleftarrow{\$} \mathcal{D}_a$
3: $y \leftarrow \mathsf{Eval}(a, x)$
4: $x' \leftarrow \mathcal{A}(a, y)$ |
5: **return** $\mathsf{Eval}(a, x') = y$

The advantage of an adversary $\mathcal{A}$ is defined as:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{TDF\text{-}OW}}() = \Pr\left[\mathsf{TDF\text{-}OWA}_{\mathcal{A}}() = 1\right]$$

where the probability is taken over the randomness of Gen, the choice of x, and $\mathcal{A}$'s randomness.

A TDF scheme is *one-way secure* if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{TDF\text{-}OW}}() \leq \mathsf{negl}().$$

3. **Trapdoor Property:** There exists a PPT algorithm $\mathcal{B}$ such that:

Game $\mathsf{TDF\text{-}TP}_{\mathcal{B}}()$**:**
1: $(a, t) \leftarrow \mathsf{Gen}()$
2: $x \xleftarrow{\$} \mathcal{D}_a$
3: $y \leftarrow \mathsf{Eval}(a, x)$
4: $x' \leftarrow \mathcal{B}(a, y, t)$
5: **return** $\mathsf{Eval}(a, x') = y$

The adversary $\mathcal{B}$ is given the trapdoor t and the output of the function f_a on a random input x. The goal of $\mathcal{B}$ is to find a preimage x' such that $\mathsf{Eval}(a, x') = y$. The trapdoor property is satisfied if the probability of $\mathcal{B}$ succeeding in this game is non-negligible. Formally, the advantage of an adversary $\mathcal{B}$ is defined as:

$$\mathsf{Adv}_{\mathcal{B}}^{\mathsf{TDF\text{-}TP}}() = \Pr\left[\mathsf{TDF\text{-}TP}_{\mathcal{B}}() = 1\right]$$

where the probability is taken over the randomness of Gen, the choice of x, and $\mathcal{B}$'s randomness.

3.1.3 Lattice Trapdoors

Recall that

$$\Lambda^{\perp}(\mathbf{A}) = \{\mathbf{z} \in \mathbb{Z}^m : \mathbf{A}\mathbf{z} = \mathbf{0} \pmod q\}$$

is a rank-m lattice. A lattice trapdoor for a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ is a short basis for the lattice $\Lambda^{\perp}(\mathbf{A})$. More generally, a set of short linearly independent vectors in $\Lambda^{\perp}(\mathbf{A})$ suffices. More explicitly:

Definition 3.1.1 (Lattice Trapdoor)**.** A matrix $\mathbf{T} \in \mathbb{Z}^{m \times m}$ is a β-good lattice trapdoor for a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ if

1. Each column vector of $\mathbf{T}$ is in the (right) mod-q kernel of $\mathbf{A}$, namely, $\mathbf{AT} = \mathbf{0} \pmod q$;
2. Each column vector of $\mathbf{T}$ is short, namely for all $i \in [m]$, $\|\mathbf{t}_i\|_\infty \leq \beta$; and
3. $\mathbf{T}$ has rank m over $\mathbb{R}$.

Note that the rank of $\mathbf{T}$ over $\mathbb{Z}_q$ can be no more than $m - n$; so, at first sight, the first and the third conditions may appear to be contradictory. However, the fact that we require the *real* rank over $\mathbf{T}$ to be large is the crucial thing here. This is related to why $\Lambda^{\perp}(\mathbf{A})$ as a lattice has rank m, even though as a linear subspace of $\mathbb{Z}_q^m$ has rank only $m - n$. Another way to look at $\mathbf{T}$ is that each of its columns is a homogeneous SIS solution with respect to $\mathbf{A}$.

3.1.4 Signature Scheme based on Lattice Trapdoors

Here is a simple digital signature scheme.

- The key generation algorithm samples a function together with a trapdoor. This would be $\mathbf{A}$ and $\mathbf{T}$. The public key is $\mathbf{A}$ and the secret key is $\mathbf{T}$.
- To sign a message m, first map it into the range of the function, e.g., by hashing it. That is, compute $\mathbf{v} = H(m)$. The signature is an inverse of $\mathbf{v}$ under the function $g_{\mathbf{A}}$. That is, a short vector $\mathbf{e}$ such that $\mathbf{Ae} = \mathbf{v} \pmod q$. This is guaranteed by the surjectivity of the function $g_{\mathbf{A}}$.
- Verification, given a message m, public key $\mathbf{A}$ and signature $\mathbf{e}$, consists of checking that $\mathbf{Ae} = H(m) \pmod q$ and that $\|\mathbf{e}\|_\infty \leq m^2$.

Unforgeability (given no signature queries) reduces to SIS in the random oracle model, i.e., assuming that H is a random oracle.

However, given signatures on adversarially chosen messages (in fact, even random messages), this scheme is broken. The key issue is that there are many inverses of $H(m)$, and the particular inverse computed using a trapdoor $\mathbf{T}$ leaks information about $\mathbf{T}$.

Collecting this leakage over sufficiently many (polynomially many) signature queries enables an adversary to find $\mathbf{T}$, allowing her to forge signatures at will going forward. The fundamental difficulty seems to stem from the fact that the inversion procedure is *deterministic*!

To mitigate the difficulty, we need a special kind of inverter: a "pre-image sampler"; that is, it is given the trapdoor $\mathbf{T}$ and produces a "random" pre-image. More precisely, we need the following distributions to be statistically close (computational indistinguishability is fine, but we will achieve statistical closeness):

$$\left(\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}, \mathbf{e} \leftarrow D_{\mathbb{Z}^m,s}, \mathbf{v} := \mathbf{Ae} \pmod q\right)$$
$$\approx_s \left(\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}, \mathbf{e} \leftarrow \mathrm{PreSamp}(\mathbf{A}, \mathbf{T}, \mathbf{v}), \mathbf{v} \leftarrow \mathbb{Z}_q^n\right)$$

3.2 Pre-Samplable Trapdoor Functions

We first define preimage sampleable (trapdoor) functions using a tuple of probabilistic polynomial-time algorithms ($\texttt{TrapGen}, \texttt{SampleDom}, \texttt{SamplePre}$). A collection of one-way preimage sampleable functions satisfies the following properties:

1. **Generating a function with trapdoor:** $\texttt{TrapGen}(1^n)$ outputs (a, t), where a describes an efficiently computable function $f_a : \mathcal{D}_n \to \mathcal{R}_n$ (for some efficiently recognizable domain $\mathcal{D}_n$ and range $\mathcal{R}_n$ depending on n), and t is the trapdoor information for f_a.

2. **Domain sampling with uniform output:** $\texttt{SampleDom}(1^n)$ samples x from some (possibly non-uniform) distribution over $\mathcal{D}_n$, such that the distribution of $f_a(x)$ is uniform over $\mathcal{R}_n$.

3. **Preimage sampling with trapdoor:** For every $y \in \mathcal{R}_n$, $\texttt{SamplePre}(t, y)$ samples from the conditional distribution of $x \sim \mathrm{SampleDom}(1^n)$, given that $f_a(x) = y$.

4. **One-wayness without trapdoor:** For any probabilistic polynomial-time algorithm $\mathcal{A}$, the probability that $\mathcal{A}(1^n, a, y) \in f_a^{-1}(y) \subseteq \mathcal{D}_n$ is negligible. The probability is taken over the choice of a, the uniform random selection of $y \in \mathcal{R}_n$, and the randomness of $\mathcal{A}$.

Trapdoor permutations (with uniform distribution over the domain) satisfy this definition since the output distribution is uniform and every point has a unique inverse.

The trapdoor functions we construct satisfy a slightly relaxed definition in which the properties hold only statistically. Specifically:

- The properties hold with overwhelming probability over the choice of a.
- $\mathrm{SampleDom}(1^n)$ outputs $x \in \mathcal{D}_n$ with overwhelming probability.

- The distribution of $f_a(x)$ is only statistically close to uniform.
- SamplePre samples from a distribution over the preimages that is statistically close to the prescribed conditional distribution.

3.3 Signature Scheme based on PSTF

The hash-and-sign paradigm for signature schemes operates as follows:

- The public verification key is a trapdoor function f.
- The signing key is the trapdoor f^{-1}.
- To sign a message m, first hash m to some point $y = H(m)$ in the range of f, then output the signature $\sigma = f^{-1}(y)$.
- To verify (m, σ), check whether $f(\sigma) = H(m)$.

3.3.1 Construction

A collection of collision-resistant PSFs can be constructed under the assumption that the SIS problem is hard on average for appropriate parameters. For security, the signer must release at most one preimage of a given point. The scheme operates relative to a function $H = H_n : \{0,1\}^* \to \mathcal{R}_n$, modeled as a random oracle (where $\mathcal{D}_n$ and $\mathcal{R}_n$ are the efficiently recognizable domain and range, respectively, for security parameter n).

- **KeyGen**(1^n): Compute $(a, t) \leftarrow \text{TrapGen}(1^n)$, where a describes a function f_a and t is its trapdoor. The public key is a, and the private key is t.
- **Sign**(t, m): If (m, σ_m) is stored locally, output σ_m. Otherwise, compute $\sigma_m \leftarrow \text{SamplePre}(t, H(m))$, store (m, σ_m), and output σ_m.
- **Verify**(a, m, σ): Accept if $\sigma \in \mathcal{D}_n$ and $f_a(\sigma) = H(m)$, otherwise reject.

3.3.2 Security Proof

Theorem 3.3.1. *The signature scheme described above is strongly existentially unforgeable under a chosen-message attack (EUF-CMA).*

Proof. We start with an assumption that the underlying hash function is collision resistant. We prove the security of this scheme via a reduction, demonstrating that if an adversary $\mathcal{A}$ can break the existential unforgeability of the signature scheme, then we can construct a poly-time adversary $\mathcal{S}$ that breaks the trapdoor collision-resistant hash function, leading to a contradiction.

Given an index a describing a function f_a, the adversary $\mathcal{S}$ runs $\mathcal{A}$ on the public key a and simulates the random oracle H and signing oracle as follows. Without loss of

generality, we assume that $\mathcal{A}$ queries H on every message m before making a signing query on m:

- **Simulation of H:** For each distinct query $m \in \{0,1\}^*$, $\mathcal{S}$ samples $\sigma_m \leftarrow \text{SampleDom}(1^n)$, stores (m, σ_m), and returns $f_a(\sigma_m)$ as $H(m)$. If H was queried previously on m, $\mathcal{S}$ simply retrieves $f_a(\sigma_m)$ from storage.
- **Simulation of the Signing Oracle:** Whenever $\mathcal{A}$ makes a signing query on m, $\mathcal{S}$ retrieves (m, σ_m) from storage and returns σ_m as the signature.

Now, assume that $\mathcal{A}$ outputs a forgery (m^*, σ^*). Since $\mathcal{A}$ is successful in breaking the scheme, we know:

1. $\sigma^* \in \mathcal{D}_n$ (i.e., the forged signature is in the domain).
2. $f_a(\sigma^*) = H(m^*)$.

Since $\mathcal{A}$ is required to query H on m^* before producing the forgery, $\mathcal{S}$ must have already sampled $\sigma_{m^*} \leftarrow \text{SampleDom}(1^n)$ and stored (m^*, σ_{m^*}) with $H(m^*) = f_a(\sigma_{m^*})$. Thus, the adversary $\mathcal{S}$ now has a collision:

$$f_a(\sigma^*) = f_a(\sigma_{m^*})$$

It remains to show that $\sigma^* \neq \sigma_{m^*}$. There are two cases to consider:

1. **Case 1:** If $\mathcal{A}$ made a signing query on m^*, it received σ_{m^*} as the signature. Since (m^*, σ^*) is considered a forgery, we must have $\sigma^* \neq \sigma_{m^*}$.
2. **Case 2:** If $\mathcal{A}$ did not make a signing query on m^*, then for the query to H on m^*, $\mathcal{S}$ stored a tuple (m^*, σ_{m^*}) where $\sigma_{m^*} \leftarrow \text{SampleDom}(1^n)$, and returned $f_a(\sigma_{m^*})$ to $\mathcal{A}$. By the preimage min-entropy property of the hash family, the min-entropy of σ_{m^*} given $f_a(\sigma_{m^*})$ (and the rest of $\mathcal{A}$'s view, which is independent of σ_{m^*}) is $\omega(\log n)$. Thus, $\sigma^* \neq \sigma_{m^*}$ except with negligible probability $2^{-\omega(\log n)}$.

We conclude that $\mathcal{S}$ outputs a valid collision in f_a with probability negligibly close to $\epsilon(n)$, contradicting the assumed collision resistance of f_a. Thus, the signature scheme is strongly existentially unforgeable under a chosen-message attack. □

3.4 Conclusion

In this chapter, we introduced the concept of trapdoor functions (TDFs) and explored their role in constructing cryptographic signature schemes. We also presented preimage sampleable trapdoor functions (PSTFs), which enable the design of secure signatures by allowing efficient sampling of valid preimages. The security of PSTFs relies on the average-case hardness of the Short Integer Solution (SIS) problem.

Motivated by the idea that a functional-level abstraction of signature schemes allows for clearer analysis and generalization, we next turn our attention to ring signatures. This will naturally lead us to the construction and study of ring trapdoor functions.

4

Ring Signatures

Ring signatures are a type of digital signature that allows a member of a group to sign a message on behalf of the group without revealing which member signed it. It varies from group signatures in that it doesn't require a trusted group manager to issue the signatures. This property is useful in scenarios where privacy and anonymity are important, such as in voting systems or whistleblowing applications. The concept of ring has since been widely studied and applied in various cryptographic protocols.

The name "ring" comes from the fact that the signature is generated using a set of public keys, which can be thought of as forming a "ring" around the signer.

4.1 Definition of Ring Signatures

4.1.1 Syntax

A *ring signature* scheme RSig [GJK24] is given by four algorithms $(\mathsf{Stp}, \mathsf{Gen}, \mathsf{Sgn}, \mathsf{Ver})$.

> $par \xleftarrow{\$} \mathsf{Stp}(\kappa)$: Given an upper bound, $\kappa > 1$, on the ring size, the probabilistic setup algorithm Stp returns system parameters par, where par defines a message space $\mathcal{M}$. We assume that all algorithms are implicitly given access to the system parameters par.
>
> $(pk, sk) \xleftarrow{\$} \mathsf{Gen}$: The probabilistic key generation algorithm returns a secret key sk and a corresponding public key pk.
>
> $\sigma \xleftarrow{\$} \mathsf{Sgn}(sk, \rho, m)$: Given a secret key sk, a ring $\rho = \{pk_1, \ldots, pk_k\}$ such that the public key pk corresponding to sk satisfies $pk \in \rho$ and $k \leq \kappa$, and a message $m \in \mathcal{M}$, the probabilistic signing algorithm Sgn returns a signature σ from a signature space $\mathcal{S}$.
>
> $b \leftarrow \mathsf{Ver}(\sigma, \rho, m)$: Given a signature σ, a ring ρ, and a message m, the deterministic verification algorithm Ver returns a bit $b \in \{0, 1\}$.

4.1.2 Properties

1. **Correctness:** RSig is $\delta(\kappa)$-*correct* or has *correctness error* $\delta(\kappa)$ if $\forall \kappa \in \mathbb{N}$, $par \xleftarrow{\$} \mathsf{Stp}(\kappa)$, and $\{(pk_i, sk_i)\}_{i\in[k]} \in \mathsf{sup}(\mathsf{Gen})$, and for any $i \in [k]$ with $k \leq \kappa$,

$$\Pr\left[\mathsf{Ver}(\mathsf{Sgn}(sk_i, \rho, m), \rho, m) \neq 1\right] \leq \delta(\kappa).$$

   where $\rho := \{pk_1, \ldots, pk_k\}$, and the probability is taken over the random choices of Stp, Gen, and Sgn. We assume (w.l.o.g.) that there is a mapping μ from the space of secret keys to the space of public keys such that for all $(pk, sk) \in \mathsf{sup}(\mathsf{Gen})$, it holds that $\mu(sk) = pk$.

2. **Unforgeability:** We will consider the strongest unforgeability notion for ring signatures, i.e., "insider security" considered in [**?**]. Given a target set of public keys $\rho = \{pk_1, pk_2, \cdots, pk_n\}$, there doesn't exist an adversary that can forge a signature σ^* on a message m^* and a ring $\rho^* \subseteq \rho$. The adversary is also allowed to make adaptive signing queries on a message m_i and ring ρ_i, as long as the ring contains at least one of the supplied keys from ρ (and hence the experiment knows the corresponding secret key which is equivalent to having a corrupt "insider" in the ring). We will consider the following two variants of unforgeability:

   (a) *Unforgeability under chosen ring attack (UF-CRA):* The adversary is allowed to make adaptive signing queries on a message m_i and ring ρ_i, as long as the ring contains at least one of the supplied keys from ρ.

   (b) *One-per-message Unforgeability under chosen ring attack (UF-CRA1):* The adversary is allowed to at most one signing query per message/ring pair (m_i, ρ_i).

   The above two notions are formalized through the games

$$(n, \kappa, Q_{\mathsf{Sgn}})\text{-UF-CRA}_{\mathrm{RSig}}(\mathcal{A}) \quad \text{and} \quad (n, \kappa, Q_{\mathsf{Sgn}})\text{-UF-CRA1}_{\mathrm{RSig}}(\mathcal{A}),$$

   depicted below, where n is the number of users, κ is the maximal ring size, and Q_{Sgn} is an upper bound on the signing queries. We define the advantage functions of adversary $\mathcal{A}$ as

$$\mathsf{Adv}^{(n,\kappa,Q_{\mathsf{Sgn}})\text{-UF-CRA}}_{\mathrm{RSig},\mathcal{A}} := \Pr[(n, \kappa, Q_{\mathsf{Sgn}})\text{-UF-CRA}_{\mathrm{RSig}}(\mathcal{A}) \to 1],$$

$$\mathsf{Adv}^{(n,\kappa,Q_{\mathsf{Sgn}})\text{-UF-CRA1}}_{\mathrm{RSig},\mathcal{A}} := \Pr[(n, \kappa, Q_{\mathsf{Sgn}})\text{-UF-CRA1}_{\mathrm{RSig}}(\mathcal{A}) \to 1].$$

Games $(n, \kappa, Q_{\mathsf{Sgn}})$-UF-CRA$_{\mathrm{RSig}}(\mathcal{A})$ and $(n, \kappa, Q_{\mathsf{Sgn}})$-UF-CRA1$_{\mathrm{RSig}}(\mathcal{A})$

1: $Q \leftarrow \emptyset$
2: $par \xleftarrow{\$} \mathsf{Stp}(\kappa)$

3: **for** $i \in [n]$ **do**
4: $(pk_i, sk_i) \xleftarrow{\$} \mathsf{Gen}$
5: **end for**
6: $(\sigma^*, \rho^*, m^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{Sgn}}(par, pk_1, \ldots, pk_n)$
7: **return** $[[\rho^* \subseteq \rho \wedge \mathsf{Ver}(\sigma^*, \rho^*, m^*) = 1 \wedge (\rho^*, m^*) \notin Q]]$

Oracle $\mathrm{Sgn}(i \in [n], \rho, m)$
1: **if** $pk_i \notin \rho$ **then**
2: **return** $\perp$
3: **end if**
4: **if** $(\rho, m) \in Q$ **then**
5: **return** $\perp$ ▷ Prevent multiple signing queries on the same message as per UF-CRA1
6: **end if**
7: $\sigma \xleftarrow{\$} \mathrm{Sgn}(sk_i, \rho, m)$
8: $Q \leftarrow Q \cup \{(\rho, m)\}$
9: **return** σ

Another notion of unforgeability (weaker) as considered in [**?**] is *Unforgeability against fixed-ring attacks (UF-FRA)*. The adversary is restricted to make signing queries with respect to the *full ring* $\rho' = \{pk_1, pk_2, \cdots, pk_\kappa\}$ and the forgery is also required to verify with respect to ρ'. The

The advantage function is defined as

$$\mathsf{Adv}^{(\kappa, Q_{\mathsf{FRASgn}})\text{-UF-FRA}}_{\mathrm{RSig}, \mathcal{A}} := \Pr[(\kappa, Q_{\mathsf{FRASgn}})\text{-UF-FRA}_{\mathrm{RSig}}(\mathcal{A}) \to 1],$$

We will define the game $(\kappa, Q_{\mathsf{FRASgn}})$-UF-FRA$_{\mathrm{RSig}}(\mathcal{A})$ in a similar manner as above.

Games $(\kappa, Q_{\mathsf{FRASgn}})$-UF-FRA$_{\mathrm{RSig}}(\mathcal{A})$
1: $Q \leftarrow \emptyset$
2: $par \xleftarrow{\$} \mathsf{Stp}(\kappa)$
3: **for** $i \in [\kappa]$ **do**
4: $(pk_i, sk_i) \xleftarrow{\$} \mathsf{Gen}$
5: **end for**
6: $(\sigma^*, m^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{FRASgn}}(par, pk_1, \ldots, pk_\kappa)$
7: **return** $[[\rho' \wedge \mathsf{Ver}(\sigma^*, m^*) = 1 \wedge m^* \notin Q]]$

Oracle $\mathrm{FRASgn}(i \in [\kappa], m)$
1: **if** $pk_i \notin \rho'$ **then**
2: **return** $\perp$

3: **end if**
4: $\sigma \xleftarrow{\$} \mathrm{Sgn}(sk_i, \rho', m)$
5: $Q \leftarrow Q \cup \{m\}$
6: **return** σ

3. **Anonymity:** We will consider the strongest anonymity notion for ring signatures, i.e., *anonymity under full key exposure* considered in [**?**]. Intutively, this means that the adversary should not be able to distinguish between the real signer and the other ring members even if the secret keys of all the ring members are exposed. The adversary is given the public keys, a signing oracle and it chooses a message and two indices (before getting the secret keys). After this, the adversary gets the secret keys of all the ring members and a challenge signature and have to guess the index of the real signer. A natural extension of this notion is *multi-challenge anonymity under full key exposure (MC-Ano)* where the adversary is given multiple challenge signatures and have to guess the indices of the real signers for all the challenges. The advantage function of the adversary is defined as

$$\mathsf{Adv}^{(n,\kappa,Q_{\mathsf{Chl}})\text{-MC-Ano}}_{\mathrm{RSig},\mathcal{A}} := \left|\Pr[(n,\kappa,Q_{\mathsf{Chl}})\text{-MC-Ano}_{\mathrm{RSig}}(\mathcal{A}) \to 1] - \frac{1}{2}\right|.$$

We will define the game $(n,\kappa,Q_{\mathsf{Chl}})$-MC-Ano$_{\mathrm{RSig}}(\mathcal{A})$ in a similar manner as above.

Game $(n,\kappa,Q_{\mathrm{Ch1}})$-MC-Ano$_{\mathrm{RSig}}(\mathcal{A})$
1: $par \xleftarrow{\$} \mathrm{Stp}(\kappa)$
2: **for** $i \in [n]$ **do**
3: $\quad (pk_i, sk_i) \xleftarrow{\$} \mathrm{Gen}$
4: **end for**
5: $b \xleftarrow{\$} \{0,1\}$
6: $b' \xleftarrow{\$} \mathcal{A}^{\mathrm{Ch1}}(par, (pk_1, sk_1), \ldots, (pk_n, sk_n))$
7: **return** $[b = b']$

Oracle $\mathrm{Ch1}(i_0 \in [n], i_1 \in [n], \rho, m)$
1: **if** $(\rho \subseteq \{pk_1, \ldots, pk_n\}) \wedge (pk_{i_0} \in \rho) \wedge (pk_{i_1} \in \rho)$ **then**
2: $\quad \sigma \xleftarrow{\$} \mathrm{Sgn}(sk_{b_i}, \rho, m)$
3: $\quad$ **return** σ
4: **else**
5: $\quad$ **return** $\perp$
6: **end if**

5

Ring Trapdoor Functions

In this chapter, we introduce the concept of *ring trapdoor functions* (RTDFs) and present a construction based on the NTRU assumption. We will see how this definition generalizes the notion of ring signatures and how it can be used to construct efficient and secure ring signature schemes. The construction is based on the NTRU assumption, which is a well-studied lattice-based assumption. We will also discuss the security properties of our construction and how they relate to the underlying assumptions.

5.1 Definition of Ring Trapdoor Functions

5.1.1 Syntax

A *ring trapdoor function* RTDF is defined by five algorithms $(\mathsf{Stp}, \mathsf{Gen}, \mathsf{Eval}, \mathsf{Inv}, \mathsf{Smp})$.

> $par \xleftarrow{\$} \mathsf{Stp}(\kappa)$: Given an upper bound, $\kappa > 1$, on the ring size, the probabilistic setup algorithm Stp returns system parameters par, where par defines the range $\mathcal{R}$. We assume that all algorithms are implicitly given access to the system parameters par.
>
> $(a, t) \xleftarrow{\$} \mathsf{Gen}()$: The probabilistic key generation algorithm returns a as the description of a function $f_a : \mathcal{D}_a \to \mathcal{R}$ and t as some trapdoor information for f_a. Here, $\mathcal{D}_a$ is the domain defined by the function descriptor a.
>
> $y \leftarrow \mathsf{Eval}(\rho = \{a_1, \cdots, a_k\}, x)$: Given a ring $\rho = \{a_1, \ldots, a_k\}$ such that $k \leq \kappa$, and an element $x \in \mathcal{D}$ where $\mathcal{D}$ is the domain specified by the ring ρ, the deterministic evaluation algorithm Eval returns $y \in \mathcal{R}$.
>
> $x \xleftarrow{\$} \mathsf{Inv}(t, \rho, y)$: Given a trapdoor t, a ring $\rho = \{a_1, \ldots, a_k\}$ such that $k \leq \kappa$, an element $y \in \mathcal{R}$, the probabilistic inverse algorithm Inv returns an element $x \in \mathcal{D}$.
>
> $x \xleftarrow{\$} \mathsf{Smp}(\rho = \{a_1, \cdots, a_k\})$: Given a ring $\rho = \{a_1, \ldots, a_k\}$ such that $k \leq \kappa$, the probabilistic sampling algorithm Smp returns an element $x \in \mathcal{D}$.

The function RTDF is formally denoted as $f_\rho : \mathcal{D} \to \mathcal{R}$, where $f_\rho(x) = \mathsf{Eval}(\rho, x)$. The inverse of the function is denoted as $f_\rho^{-1} : \mathcal{R} \to \mathcal{D}$, where $f_\rho^{-1}(y) = \mathsf{Inv}(t, \rho, y)$. Let μ be a function that maps a function description to its corresponding trapdoor: $\mu(a) = t$.

5.1.2 Properties

1. **Correctness:** RTDF is $\delta(\kappa)$-*correct* or has *correctness error* $\delta(\kappa)$ if $\forall \kappa \in \mathbb{N}$, $par \xleftarrow{\$} \mathsf{Stp}(\kappa)$, and $\{(a_i, t_i)\}_{i\in[k]} \in \mathsf{sup}(\mathsf{Gen})$, and for any $i \in [k]$ with $k \leq \kappa$,

$$\Pr\left[f_\rho^{-1}(f_\rho(x)) \neq x\right] \leq \delta(\kappa).$$

   where $\rho := \{a_1, \ldots, a_k\}$, and the probability is taken over the random choices of Stp, Gen, and Inv.

2. **One-wayness without trapdoor:** For any probabilistic adversary $\mathcal{A}$, the advantage function of $\mathcal{A}$ in breaking the one-wayness of RTDF is defined as:

$$\mathsf{Adv}^{\mathsf{OW}(\kappa,k)}_{\mathsf{RTDF},\mathcal{A}} := \max_k \left(\Pr\left[\mathsf{OW}^{\mathsf{RTDF}}_{\mathcal{A}}(\kappa, k) = 1\right]\right),$$

   where the security experiment $\mathsf{OW}^{\mathsf{RTDF}}_{\mathcal{A}}(\kappa, k)$ is defined as follows:

Game $\mathsf{OW}^{\mathsf{RTDF}}_{\mathcal{A}}(\kappa, k)$

1: $par \xleftarrow{\$} \mathsf{Stp}(\kappa)$
2: **for** $i \in [k]$ **do**
3: $\quad (a_i, t_i) \xleftarrow{\$} \mathsf{Gen}()$
4: **end for**
5: $\rho \leftarrow \{a_1, \ldots, a_k\}$
6: $y \xleftarrow{\$} \mathcal{R}$
7: $x' \xleftarrow{\$} \mathcal{A}(par, \rho, y)$
8: **return** 1 if $\mathsf{Eval}(\rho, x') = y$, else 0

   The probability is taken over $par \xleftarrow{\$} \mathsf{Stp}(\kappa)$, $(a_i, t_i) \xleftarrow{\$} \mathsf{Gen}()$, $y \xleftarrow{\$} \mathcal{R}$, and $\mathcal{A}$'s randomness.

3. **Domain Sampling:** Let Q_{Smp} be the maximum number of sampling queries. For any probabilistic adversary $\mathcal{A}$, the advantage function of $\mathcal{A}$ in breaking the domain sampling security of RTDF is defined as:

$$\mathsf{Adv}^{\mathsf{DS}(n,\kappa)}_{\mathsf{RTDF},\mathcal{A}} := \left|\Pr\left[\mathsf{DS}^{\mathsf{RTDF}}_{\mathcal{A}}(n, \kappa) = 1\right] - \frac{1}{2}\right|$$

   The security experiment $\mathsf{DS}^{\mathsf{RTDF}}_{\mathcal{A}}(n, \kappa)$ is defined as follows:

Game $\mathsf{DS}_{\mathcal{A}}^{\mathsf{RTDF}}(n,\kappa)$

1: $par \xleftarrow{\$} \mathsf{Stp}(\kappa)$
2: **for** $i \in [n]$ **do**
3: $\quad (a_i, t_i) \xleftarrow{\$} \mathsf{Gen}()$
4: **end for**
5: $b \xleftarrow{\$} \{0,1\}$
6: $b' \xleftarrow{\$} \mathcal{A}^{\mathsf{Sample}}(par, \{a_1, \ldots, a_n\})$
7: **return** $[b' = b]$

Oracle $\mathsf{Sample}(\rho = \{a_{i_1}, \ldots, a_{i_k}\})$

1: **if** $k > \kappa$ **or** $\exists a(a \in \rho \wedge a \notin \{a_1, \ldots, a_n\})$ **then**
2: $\quad$ **return** $\perp$
3: **end if**
4: **if** $b = 0$ **then**
5: $\quad x \xleftarrow{\$} \mathsf{Smp}(\rho)$
6: $\quad y \leftarrow \mathsf{Eval}(\rho, x)$
7: **else**
8: $\quad y \xleftarrow{\$} \mathcal{R}$
9: **end if**
10: **return** y

The probability is over $par \xleftarrow{\$} \mathsf{Stp}(\kappa)$, $(a_i, t_i) \xleftarrow{\$} \mathsf{Gen}()$, $b \xleftarrow{\$} \{0,1\}$, $\mathcal{A}$'s randomness, and the responses from Sample.

4. **Preimage Sampling:** For any probabilistic adversary $\mathcal{A}$, the advantage function of $\mathcal{A}$ in distinguishing the output of Inv from the conditional distribution of Smp is defined as:

$$\mathsf{Adv}_{\mathsf{RTDF},\mathcal{A}}^{\mathsf{PS}(n,\kappa)} := \left| \Pr\left[\mathsf{PS}_{\mathcal{A}}^{\mathsf{RTDF}}(n,\kappa) = 1\right] - \frac{1}{2} \right|,$$

where the security experiment $\mathsf{PS}_{\mathcal{A}}^{\mathsf{RTDF}}(n,\kappa)$ is as follows:

Game $\mathsf{PS}_{\mathcal{A}}^{\mathsf{RTDF}}(n,\kappa)$

1: $par \xleftarrow{\$} \mathsf{Stp}(\kappa)$
2: **for** $i \in [n]$ **do**
3: $\quad (a_i, t_i) \xleftarrow{\$} \mathsf{Gen}()$
4: **end for**
5: $b \xleftarrow{\$} \{0,1\}$
6: $b' \xleftarrow{\$} \mathcal{A}^{\mathsf{Challenge}}(par, \{a_1, \ldots, a_n\})$
7: **return** $[b' = b]$

Oracle $\mathsf{Challenge}(\rho = \{a_{i_1}, \ldots, a_{i_k}\}, y \in \mathcal{R})$
1: **if** $k > \kappa$ **or** $\exists a(a \in \rho \wedge a \notin \{a_1, \ldots, a_n\})$ **then**
2: **return** $\perp$
3: **end if**
4: **if** $b = 0$ **then**
5: $t \xleftarrow{\$} \{\mu(a) \mid a \in \rho\}$
6: $x \xleftarrow{\$} \mathsf{Inv}(t, \rho, y)$
7: **else**
8: Sample $x \xleftarrow{\$} \mathsf{Smp}(\rho)$ until $f_\rho(x) = y$
9: **end if**
10: **return** x

The probability is over $par \xleftarrow{\$} \mathsf{Stp}(\kappa)$, $(a_i, t_i) \xleftarrow{\$} \mathsf{Gen}()$, $b \xleftarrow{\$} \{0, 1\}$, $\mathcal{A}$'s randomness, and $\mathsf{Challenge}$'s responses.

5. **Anonymity:** For any probabilistic polynomial-time adversary $\mathcal{A}$, given a ring $\{a_1, \cdots, a_n\}$ with it's trapdoors $\{t_1, \cdots, t_n\}$ and an inverse oracle OInv, chooses an element $x \in \mathcal{D}$ and two indices. After this, the adversary gets a challenge output y and have to guess the index of the original element. The advantage function of the adversary is defined as

$$\mathsf{Adv}^{(n,\kappa,Q_{\mathsf{OInv}})\text{-MC-Ano}}_{\mathrm{RTDF},\mathcal{A}} := \left| \Pr[(n, \kappa, Q_{\mathsf{OInv}})\text{-MC-Ano}_{\mathrm{RTDF}}(\mathcal{A}) \to 1] - \frac{1}{2} \right|.$$

We will define the game $(n, \kappa, Q_{\mathsf{OInv}})$-MC-Ano$_{\mathrm{RTDF}}(\mathcal{A})$, where n is the number of users, κ is the maximal ring size, and Q_{OInv} is an upper bound on the challenges. For simplicity, we can restrict the adversary to only query the inverse oracle on the elements of the ring. The game is defined as follows:

Game $(n, \kappa, Q_{\mathrm{OInv}})$-MC-Ano$_{\mathrm{RTDF}}(\mathcal{A})$
1: $par \xleftarrow{\$} \mathrm{Stp}(\kappa)$
2: **for** $i \in [n]$ **do**
3: $(a_i, t_i) \xleftarrow{\$} \mathrm{Gen}$
4: **end for**
5: $b \xleftarrow{\$} \{0, 1\}$
6: $b' \xleftarrow{\$} \mathcal{A}^{\mathrm{OInv}}(par, (a_1, t_1), \ldots, (a_n, t_n))$
7: **return** $[b = b']$

Oracle $\mathrm{OInv}(i_0 \in [n], i_1 \in [n], \rho = \{a_1, \ldots, a_{k \le n}\}, x)$
1: **if** $(\rho \subseteq \{a_1, \ldots, a_n\}) \wedge (a_{i_0} \in \rho) \wedge (a_{i_1} \in \rho)$ **then**
2: $y \leftarrow \mathrm{Eval}(\rho, x)$

```
    return y
else
    return ⊥
end if
```

5.2 Construction: NTRU-Based Ring Trapdoor Function

Our ring trapdoor function construction is defined over R_q and is instantiated with the trapdoor generation algorithm TpdGen (as described in `ANTRAG` [ENS+23]) and the preimage sampling algorithm PreSmp (as described in `MITAKA` [EFG+22]).

The setup algorithm Stp takes the maximal ring size κ and outputs the parameters $par = (\kappa, \tau, \beta)$, where τ is the tail-cut parameter for Gaussian sampling, and β is the norm bound. The function ψ sets an appropriate tailcut rate τ based on κ. For brevity, we omit general parameters (e.g., ring dimension n, modulus q, Gaussian width s, trapdoor quality α) in Stp.

The domain $\mathcal{D}$ consists of $(k+1)$-tuples $(u_1, u_2, \cdots, u_k, v) \in R^{k+1}$ with ℓ_2 norm of each element of R is bounded by β, where $R = \frac{\mathbb{Z}[x]}{(x^n+1)}$. The range $\mathcal{R}$ is the quotient ring $R_q = \frac{\mathbb{Z}_q[x]}{(x^n+1)}$.

The trapdoor generation algorithm TpdGen samples (f, g) from a discrete Gaussian distribution over R, computes $h \equiv g \cdot f^{-1} \mod q$, and outputs:

- A function description $a = h$, defining $f_h : R^2 \mapsto R_q$ such that $f(u, v) = h \cdot u + v \mod q$,
- A trapdoor $t = (f, g)$.

The evaluation algorithm Eval computes $F_\rho(u_1, u_2, \cdots, u_k, v)$ where $\rho = (h_1, h_2, \cdots, h_k)$. The inversion algorithm Inv uses (f_j, g_j) to sample a preimage for $y \in R_q$. The sampling algorithm Smp samples elements from the domain.

The construction is based on the NTRU assumption, and the security of the ring trapdoor function relies on the hardness of the underlying lattice problem.

$\mathsf{Stp}(\kappa)$

1: $\tau \leftarrow \psi(\kappa)$
2: $\beta \leftarrow \tau \cdot s \cdot \sqrt{(\kappa+1)n}$
3: $par \leftarrow (\kappa, \tau, \beta) \in \mathbb{N} \times \mathbb{N} \times \mathbb{R}$
4: **return** par

$\mathsf{Gen}()$

1: $(f, g, h) \xleftarrow{\$} \mathsf{TpdGen}(q, \alpha)$
2: $a \leftarrow h$
3: $t \leftarrow (f, g)$
4: **return** (a, t)

$\mathsf{Eval}(\rho = \{h_1, h_2, \cdots, h_k\}, x = (u_1, u_2, \cdots, u_k, v) \in \mathcal{D})$

1: $y \leftarrow \sum_{i=1}^{k} h_i u_i + v \mod q \in R_q$
2: **return** y

$\mathsf{Inv}(t_j = (f_j, g_j), \rho = \{h_1, h_2, \cdots, h_k\}, y)$

1: Require $y \in R_q$
2: Require $k \leq \kappa$
3: Require $\exists j \in [k] : \mu(t_j) \in \rho$
4: **for** $i \in [k] \setminus \{j\}$ **do**
5: $\quad u_i \xleftarrow{\$} D_{\mathcal{D}_i, s}$
6: **end for**
7: $(u_j, v) \xleftarrow{\$} \mathsf{PreSmp}(B_{f_j, g_j}, s, y - \sum_{i \neq j} h_i u_i)$
8: $x = (u_1, u_2, \cdots, u_k, v)$
9: **return** x

$\mathsf{Smp}(\rho = \{h_1, h_2, \cdots, h_k\})$

1: $x \xleftarrow{\$} D_{R,s}^{k+1}$
2: **return** x

5.2.1 Security Analysis

1. **Correctness**: The RTDF construction has correctness error $\delta(\kappa)$ if PreSmp succeeds with overwhelming probability. Formally, $\forall \kappa \in \mathbb{N}$, $par \leftarrow \mathsf{Stp}(\kappa)$, $\{(a_i, t_i)\} \leftarrow \mathsf{Gen}^k()$,

$$\Pr\left[f_\rho^{-1}(f_\rho(x)) \neq x\right] \leq \delta(\kappa)$$

Proof. Let $x = (u_1, ..., u_k, v)$ be sampled via Smp. The inversion succeeds if for some $y \in R_q$:

$$\mathsf{Inv}(t_j, \rho, y) \to (u_1', ..., u_k', v')$$
$$\text{s.t. } \sum_{i=1}^{k} h_i u_i' + v' \equiv \sum_{i=1}^{k} h_i u_i + v \mod q$$

From PreSmp's specification, for the chosen j:

$$(f_j, g_j) \text{ allows solving } h_j u_j' + v' \equiv y - \sum_{i \neq j} h_i u_i' \mod q$$

The failure probability comes from:

- Gaussian tail bounds: $\Pr[\|x\| > \beta] \leq e^{-\pi\tau^2}$ via parameter choice
- PreSmp failure: $\leq 2^{-n}$ as per [EFG+22]

Thus $\delta(\kappa) \leq (k+1)e^{-\pi\tau^2} + 2^{-n} = \mathsf{negl}(\kappa)$. □

2. **One-Wayness under NTRU Assumption**: For any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ against the one-wayness of the RTDF construction, there exists an NTRU distinguisher $\mathcal{D}$ such that:

$$\mathsf{Adv}^{\mathsf{OW}}_{\mathsf{RTDF},\mathcal{A}}(\kappa, k) \leq \mathsf{Adv}^{\mathsf{NTRU}}_{\mathcal{D}}(\kappa) + \mathsf{negl}(\kappa)$$

where q is the modulus and n the lattice dimension.

Proof. We construct an NTRU distinguisher $\mathcal{D}$ that simulates the one-wayness game:

NTRU Distinguisher $\mathcal{D}$

Receive challenge (h^*, y^*) where $y^* = h^*u + v$ (NTRU sample) or $y^* \xleftarrow{\$} R_q$ (random)
Generate public parameters $\rho = \{h_1, ..., h_{k-1}, h^*\}$ with $(h_i, t_i) \leftarrow \mathsf{Gen}()$
Run $\mathcal{A}(par, \rho, y^*)$ to obtain candidate preimage $x' = (u'_1, ..., u'_k, v')$
if $\sum_{i=1}^{k} h_i u'_i + v' \equiv y^* \mod q$ **then**
    if $h^*u'_k + v' \equiv y^* \mod q$ **then**
        return 1 (NTRU)
    else
        return 0 (random)
    end if
else
    return random bit $b \xleftarrow{\$} \{0, 1\}$
end if

Analysis:

- When y^* is an NTRU sample, $\mathcal{A}$'s view perfectly simulates the real one-wayness game
- When y^* is random, $\mathcal{A}$ succeeds only if collisions exist in R_q, which is negligible

Thus $\mathcal{D}$'s distinguishing advantage satisfies $\mathsf{Adv}^{\mathsf{NTRU}}_{\mathcal{D}} \geq \mathsf{Adv}^{\mathsf{OW}}_{\mathcal{A}} - \mathsf{negl}(\kappa)$. □

3. **Domain Sampling Indistinguishability:** For any PPT adversary $\mathcal{A}$ making Q evaluation queries, the advantage in distinguishing real domain samples from random satisfies:

$$\mathsf{Adv}^{\mathsf{DS}}_{\mathsf{RTDF},\mathcal{A}}(n, \kappa) \leq Q\left(\mathsf{Adv}^{\mathsf{NTRU}} + \frac{1}{2}e^{-\pi s^2}\right)$$

where s is the Gaussian width parameter.

The security stems from two complementary properties:

- *NTRU Hardness*: Individual product terms $h_i u_i$ in $y = \sum h_i u_i + v$ look random without trapdoors
- *Gaussian Blinding*: The noise v statistically "smoothes" any residual structure through its exponential tail decay

We gradually replace real computations with random values:

- $\mathbf{H_0}$: Real world $y = h_1 u_1 + \cdots + h_k u_k + v$
- $\mathbf{H_j}$: First j terms random $y = r_1 + \cdots + r_j + h_{j+1} u_{j+1} + \cdots + v$
- $\mathbf{H_Q}$: Full randomization $y = r_1 + \cdots + r_k + v$

Between adjacent hybrids $\mathbf{H_{j-1}}$ and $\mathbf{H_j}$:

- *Computational Step*: Distinguishing $h_j u_j$ from random r_j reduces directly to NTRU assumption. If adversaries could spot this change, they could break NTRU's pseudorandomness.
- *Statistical Step*: The Gaussian noise v masks any "leftover" structure with exponentially small error. Specifically, the smoothing lemma guarantees that two distributions become statistically indistinguishable when $s > \eta_\epsilon(\Lambda)$, the lattice smoothing parameter.

Each hybrid transition contributes at most $\mathsf{Adv}^{\mathsf{NTRU}} + \frac{1}{2} e^{-\pi s^2}$ advantage. After Q queries, the total advantage grows linearly but remains negligible under proper parameter choice.

Intuitively, each use of the sampler either "looks" like an NTRU-style product (computationally) or is hidden by the Gaussian's negligible tail (statistically), and so even over Q queries the adversary gains at most Q times these small quantities.

4. **Preimage Sampling Indistinguishability:** For any PPT adversary $\mathcal{A}$ making Q_{chal} challenge queries, the advantage in distinguishing the outputs of the inversion algorithm Inv from the sampling algorithm Smp is bounded by the number of queries and the NTRU assumption.

5. **Anonymity:** Even with knowledge of all trapdoors, for any PPT adversary $\mathcal{A}$ making Q queries:
$$\mathsf{Adv}^{\mathsf{Ano}}_{\mathsf{RTDF},\mathcal{A}}(n, \kappa, Q) \leq \mathsf{negl}(\kappa)$$

Proof. The adversary's advantage in distinguishing between two trapdoor functions f_{i_0}, f_{i_1} is negligible. The proof relies on the following observations:

- The inversion equations $\sum h_j u_j^{(b)} + v^{(b)} = y$ for $b \in \{0, 1\}$ are equally satisfiable
- Gaussian re-randomization: For any u_j, set $u'_j = u_j + \delta$ with $\delta \sim D_{R,s}$ preserves distribution

- All (f_i, g_i) pairs have covariance $\leq s^2 I$, making different trapdoors statistically indistinguishable

The adversary's advantage arises only from hash collisions in R_q, which is negligible. The trapdoor functions are indistinguishable even with knowledge of all trapdoors. Thus, the adversary's advantage in distinguishing between two trapdoor functions is negligible. □

In the next section, we will show how to construct a ring signature scheme using the ring trapdoor function framework. The construction will be based on the NTRU assumption and will inherit the security properties of the underlying RTDF. But first, let's discuss the motivation behind introducing ring trapdoor functions and how they relate to existing cryptographic primitives.

5.2.2 Why Ring Trapdoor Functions?

We introduce an additional level of abstraction from ring signatures for the following reasons:

1. It enables a deeper and more fine-grained understanding of the underlying primitives.

2. Improvements at the primitive level propagate to ring signatures. For example, increasing the number of users in a ring can be done in a black-box manner without compromising security—something not possible at the ring signature level.

3. As a low-level primitive, RTDF serves as a foundation upon which other cryptographic primitives can be built.

A literature survey reveals that a similar abstraction has been previously explored in the context of ring signatures, notably in [BK10]. That work introduced a definition of ring trapdoor functions and demonstrated that it implies ring signatures. However, it is important to note that their definition is less flexible than the one proposed here. The RTDF framework enables a more general formulation of ring trapdoor functions, capable of capturing a broader class of constructions, including those not supported by the BK10 framework.

Consider the following definition of ring trapdoor functions from [BK10]:

Ring trapdoor functions. A family of (one-way) *ring trapdoor functions* is a collection of functions $\mathcal{T} = \{\mathcal{T}_\lambda\}_{\lambda\in\mathbb{N}}$, where $\mathcal{T}_\lambda = \{f : X_\lambda \to G_\lambda\}$, $X = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ is a collection of efficiently recognizable sets, and $G = \{G_\lambda\}_{\lambda\in\mathbb{N}}$ is a collection of commutative groups, where group operations can be performed efficiently, such that

1. **Sampling**: Given 1^λ, we can sample $f \in \mathcal{T}_\lambda$.

2. **Zero**: For every $\lambda \in \mathbb{N}$ there exists an efficiently recognizable element $\xi_\lambda \in X_\lambda$, such that for all $f \in \mathcal{T}_\lambda$ it holds that $f(\xi_\lambda) = 0$.

3. **Verifiability**: For every $\lambda \in \mathbb{N}$ and every polynomial k, given any $f_1, \ldots, f_k \in \mathcal{T}_\lambda$, any $x_1, \ldots, x_k \in X_\lambda$, and any $y \in G_\lambda$, one can efficiently verify that $\sum_{i\in[k]} f_i(x_i) = y$.

4. **Ring one-way**: For every polynomial k, given $f_1, \ldots, f_k \leftarrow \mathcal{T}_\lambda$ and $y \xleftarrow{\$} G_\lambda$, it is computationally hard to find $x_1, \ldots, x_k \in X_\lambda$ such that $\sum_{i\in[k]} f_i(x_i) = y$. Formally, for any polynomial time adversary $\mathcal{A}$, any polynomial t, it holds that

$$\mathrm{Ring}_\lambda^{\mathrm{InvT}}\mathrm{Adv}[\mathcal{A}] = \Pr\left[\sum_{i\in[k]} f_i(x_i) = y : \begin{array}{l} f_1, \ldots, f_k \leftarrow \mathcal{T}_\lambda, y \xleftarrow{\$} G_\lambda, \\ (x_1, \ldots, x_k) \leftarrow \mathcal{A}(1^\lambda, f_1, \ldots, f_k, y) \end{array}\right] = \mathrm{negl}(\lambda).$$

5. **Trapdoor**: Given 1^λ, sample a function-trapdoor pair (f, td) such that the marginal distribution of f is statistically indistinguishable from $\mathcal{T}_\lambda$, and such that the following holds: For any polynomial k, given any $f_1, \ldots, f_k \in \mathcal{T}_\lambda$ such that td_1 is a trapdoor for f_1, and given any $y \in G_\lambda$, one can efficiently sample $x_1, \ldots, x_k \in X_\lambda$ such that $\sum_{i\in[k]} f_i(x_i) = y$. Furthermore, letting td_2 be a trapdoor for f_2, using td_2 instead of td_1 will result in a statistically indistinguishable distribution of $(\mathrm{td}_1, \mathrm{td}_2, x_1, \ldots, x_t)$.

We denote the definition from [BK10] as BK10 and the one introduced in this work as RTDF. The BK10 definition of ring trapdoor functions, while foundational, imposes structural constraints that limit its generality and fails to capture constructions like `GANDALF`.

In BK10, ring operations involve sampling k functions $f_1, \ldots, f_k$ from a collection $\mathcal{T}_\lambda$ and computing the output as:

$$\sum_{i=1}^{k} f_i(x_i) = y,$$

where each x_i is sampled independently from a generic domain X_λ. This approach introduces the following limitation: Evaluation occurs at the level of individual functions and is then aggregated. In contrast, constructions like `GANDALF` require a single function F_ρ defined over a ring ρ of size k. The BK10 framework cannot support this, as it treats functions separately with descriptors a_i and trapdoors t_i. The RTDF framework abstracts evaluation to the input level, allowing the function $F_\rho(x_1, \ldots, x_k) = \sum_{i=1}^{k} h_i \cdot u_i + v \bmod q$. This generalization encompasses both the RTDF and BK10 notions.

Furthermore, while BK10 allows functions that are not efficiently computable but require efficient verification, RTDF defines a single function F over the ring that must be both efficiently computable and verifiable. However, it does not require each individual component function to be efficiently computable or verifiable. This distinction allows for

a more general formulation of the ring trapdoor function. As a result, RTDF captures a broader class of constructions, including those not feasible under BK10, such as GANDALF.

5.3 Ring Trapdoor Functions ⇒ Signature Scheme

Let $\mathsf{RTDF} = (\mathsf{Stp}, \mathsf{Gen}, \mathsf{Eval}, \mathsf{Inv}, \mathsf{Smp})$ be a ring trapdoor function with domain $\mathcal{D}$ and range $\mathcal{R}$, where $k \leq \kappa$ for the given maximal ring size κ. Let $H : \{0,1\}^* \to \mathcal{R}$ be a hash function. We construct the ring signature scheme $\mathsf{RSig} = (\mathsf{Stp}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ as follows:

Stp(κ)
1: $par \xleftarrow{\$} \mathsf{RTDF.Stp}(\kappa)$
2: **return** par

KeyGen()
1: $par \xleftarrow{\$} \mathsf{Stp}(\kappa)$
2: $(a, t) \xleftarrow{\$} \mathsf{RTDF.Gen}()$
3: $\mathsf{pk} := a$, $\mathsf{sk} := t$
4: **return** $(\mathsf{pk}, \mathsf{sk})$

Sign(sk, ρ, m)
1: **Input:** Trapdoor $\mathsf{sk} = t_i$ for some $a_i \in \rho = \{a_1, \ldots, a_k\}$
2: $y \leftarrow H(m)$
3: $x \xleftarrow{\$} \mathsf{RTDF.Inv}(t_i, \rho, y)$
4: **return** $\sigma := x$

Verify(ρ, m, σ)
1: $y \leftarrow H(m)$
2: $y' \leftarrow \mathsf{RTDF.Eval}(\rho, \sigma)$
3: **return** 1 if $y' = y$, else 0

Correctness: For all $\kappa \in \mathbb{N}$, all rings $\rho = \{a_1, \ldots, a_k\}$ with $k \leq \kappa$, all messages $m \in \{0,1\}^*$, and all $(pk, sk) \leftarrow \mathsf{KeyGen}()$ where $pk \in \rho$, we have:

$$\Pr\left[\mathsf{Verify}(\rho, m, \mathsf{Sign}(sk, \rho, m)) = 1\right] = 1$$

5.3.1 GANDALF: Ring Signature Construction from Lattices

The ring signature construction GANDALF is defined over R_q and instantiated with a trapdoor generation algorithm `TpdGen` and a preimage sampler `PreSmp`. The setup algorithm `Stp` takes as input the maximum ring size κ and outputs the system parameters. The function ψ sets an appropriate tailcut rate τ based on κ. Function $H : \{0,1\}^* \to \mathcal{R}_q$.

Stp(κ)

$\tau := \psi(\kappa)$
$\beta := \tau \cdot s \cdot \sqrt{(\kappa+1)N}$
$par := (\kappa, \tau, \beta) \in \mathbb{N} \times \mathbb{R} \times \mathbb{R}$
return par

Gen

$(f, g, h) \xleftarrow{\$} \texttt{TpdGen}(q, \alpha)$
$sk := (f, g) \in R_q \times R_q$
$pk := h \in R_q$
return (sk, pk)

Sgn(sk, ρ, m)

parse $sk \to (f, g)$
parse $\rho \to (h_1, \ldots, h_\kappa)$
require $k \leq \kappa$
require $\exists j \in [k] : \mu(sk) = h_j$
for $i \in [k] \setminus \{j\}$ **do**
    $u_i \leftarrow^{\$} D_{\mathbb{Z}_N, s, 0}$
    $c_i := u_i * h_i \in R_q$
end for
$h := H(m, \rho) \in R_q$
$c_j := h - \sum_{i \in [k] \setminus \{j\}} c_i$
$(u_j, v) \leftarrow^{\$} \texttt{PreSmp}(B_{f,g,s}, c_j)$
$\sigma := (u_1, \ldots, u_k, v)$
return σ

Ver(σ, ρ, m)

parse $\sigma \to (u_1, \ldots, u_k, v)$
parse $\rho \to (h_1, \ldots, h_k)$
$v := H(m, \rho) - \sum_{i \in [k]} u_i * h_i$
if $\|(u_1, \ldots, u_k, v)\|_2 \leq \beta$ **then**
    return 1
else
    return 0
end if

Correctness: The ring signature scheme GANDALF is $\delta(\kappa)$-correct where

$$\delta(\kappa) = \tau^{(\kappa+1)N} \cdot e^{\frac{(\kappa+1)N}{2}(1-\tau^2)},$$

with $\tau > 1$.

Proof. For $i \in [k]$ and $k \leq \kappa$ let $(\mathsf{sk}_i, \mathsf{pk}_i) \in \sup(\mathsf{Gen})$, $\rho := \{\mathsf{pk}_1, \ldots, \mathsf{pk}_k\}$, and $\tau > 1$. We have:

$$\begin{aligned}
\Pr[\mathsf{Ver}(\mathsf{Sgn}(\mathsf{sk}_i, \rho, m), \rho, m) \neq 1] &= \Pr[\|(\mathbf{u}_1, \ldots, \mathbf{u}_k, \mathbf{v})\|_2 > \beta] \\
&= \Pr[\|(\mathbf{u}_1, \ldots, \mathbf{u}_k, \mathbf{v})\|_2 > \tau s \sqrt{(\kappa+1) \cdot N}] \\
&< \tau^{(\kappa+1)N} \cdot e^{\frac{(\kappa+1)N}{2}(1-\tau^2)}.
\end{aligned}$$

□

The scheme is unforgeabile under the UF-CMA security model described previously. The security of the scheme relies on the hardness of the underlying lattice problem, specifically the Ring-LWE and Ring-SIS problems. The security of the scheme is also based on the assumption that the hash function H is a random oracle, which is a common assumption in cryptographic constructions. For anonymity, the scheme is designed to ensure that the identity of the signer remains hidden and is bounded by the number of challenge queries made by the adversary.

Bibliography

[AMBB+13] Carlos Aguilar-Melchor, Slim Bettaieb, Xavier Boyen, Laurent Fousse, and Philippe Gaborit. Adapting lyubashevsky's signature schemes to the ring signature setting. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *AFRICACRYPT 13: 6th International Conference on Cryptology in Africa*, volume 7918 of *Lecture Notes in Computer Science*, pages 1–25, Cairo, Egypt, June 22–24 2013. Springer, Heidelberg, Germany.

[AOS02] Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. 1-out-of-n signatures from a variety of keys. In Yuliang Zheng, editor, *Advances in Cryptology - ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 415–432, Queenstown, New Zealand, December 1–5 2002. Springer, Heidelberg, Germany.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432, Warsaw, Poland, May 4–8 2003. Springer, Heidelberg, Germany.

[BK10] Zvika Brakerski and Yael Tauman Kalai. A framework for efficient signatures, ring signatures and identity based encryption in the standard model. Cryptology ePrint Archive, Paper 2010/086, 2010.

[BKP20] Ward Beullens, Shuichi Katsumata, and Federico Pintore. Calamari and falafl: Logarithmic (linkable) ring signatures from isogenies and lattices. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 464–492, Daejeon, South Korea, December 7–11 2020. Springer, Heidelberg, Germany.

[BLO18] Dan Boneh, Vadim Lyubashevsky, and Chris Others. Efficient and secure lattice constructions. In *Advances in Cryptology - CRYPTO 2018*, volume 10993 of *Lecture Notes in Computer Science*. Springer, Heidelberg, Germany, 2018.

[BR04] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Report 2004/331, 2004. `https://eprint.iacr.org/2004/331`.

[BSW02] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*. Springer, Heidelberg, Germany, 2002.

[CVH91] David Chaum and Eugène Van Heyst. Group signatures. In *Advances in Cryptology—EUROCRYPT'91: Workshop on the Theory and Application of Cryptographic Techniques Brighton, UK, April 8–11, 1991 Proceedings 10*, pages 257–265. Springer, 1991.

[EFG+22] Thomas Espitau, Pierre-Alain Fouque, François Gérard, Mélissa Rossi, Akira Takahashi, Mehdi Tibouchi, Alexandre Wallet, and Yang Yu. Mitaka: A simpler, parallelizable, maskable variant of falcon. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part III*, volume 13277 of *Lecture Notes in Computer Science*, pages 222–253, Trondheim, Norway, May 2022. Springer.

[ENS+23] Thomas Espitau, Thi Thu Quyen Nguyen, Chao Sun, Mehdi Tibouchi, and Alexandre Wallet. Antrag: Annular NTRU trapdoor generation – making mitaka as secure as falcon. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023, Part VII*, volume 14444 of *Lecture Notes in Computer Science*, pages 3–36, Guangzhou, China, December 2023. Springer.

[ESS+19] Muhammed F. Esgin, Ron Steinfeld, Amin Sakzad, Joseph K. Liu, and Dongxi Liu. Short lattice-based one-out-of-many proofs and applications to ring signatures. In Robert H. Deng, Valérie Gauthier-Uma na, Martín Ochoa, and Moti Yung, editors, *ACNS 19: 17th International Conference on Applied Cryptography and Network Security*, volume 11464 of *Lecture Notes in Computer Science*, pages 67–88. Springer, Heidelberg, Germany, June 5–7 2019.

[GJK24] Phillip Gajland, Jonas Janneck, and Eike Kiltz. Ring signatures for deniable akem: Gandalf's fellowship. In *Annual International Cryptology Conference*, pages 305–338. Springer, 2024.

[LAZ19] Xingye Lu, Man Ho Au, and Zhenfei Zhang. Raptor: A practical lattice-based (linkable) ring signature. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 2019: 17th International Conference on Applied Cryptography and Network Security*, volume 11464 of *Lecture Notes in Computer Science*, pages 110–130, Bogota, Colombia, June 2019. Springer, Heidelberg, Germany.

[LLNW16] Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. Zero-knowledge arguments for lattice-based accumulators. In *Advances in Cryptology - EUROCRYPT 2016*, volume 9666 of *Lecture Notes in Computer Science*, pages 1–31. Springer, Heidelberg, Germany, 2016.

[LNS21] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. Smile: Set membership from ideal lattices with applications to ring signatures. In *Advances in Cryptology - CRYPTO 2021*, volume 12826 of *Lecture Notes in Computer Science*, pages 611–640. Springer, Heidelberg, Germany, 2021.

[Nao02] Moni Naor. Deniable ring authentication. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 481–498, Santa Barbara, CA, USA, August 2002. Springer, Heidelberg, Germany.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC '90)*, pages 387–394, New York, NY, USA, 1990. ACM.

[RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. *Advances in Cryptology — ASIACRYPT 2001*, pages 552–565, 2001.

[YEL+21] Tsz Hon Yuen, Muhammed F. Esgin, Joseph K. Liu, Man Ho Au, and Zhimin Ding. Dualring: Generic construction of ring signatures with efficient instantiations. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021*, pages 251–281, Cham, 2021. Springer International Publishing.

Appendix A

Variants of Signatures

Let us overview some other signature scheme variants that are similar to ring signatures. In this chapter, we will look at group signatures and threshold signatures.

A.1 Group Signatures

In the group signature setting introduced by [CVH91], a group of users can sign messages on behalf of the group. The signature does not reveal the identity of the signer, but it can be verified that the signature was generated by a member of the group. The group manager can open the signature and reveal the identity of the signer. Associated to the group is a single signature-verification key *gpk* called the group public key. Each group member i has its own secret signing key based on which it can produce a signature relative to *gpk*. The core requirements are that the group manager has a secret key *gmsk* based on which it can, given a signature σ, extract the identity of the group member who created σ (traceability) and on the other hand an entity not holding *gmsk* should be unable, given a signature σ, to extract the identity of the group member who created σ (anonymity).

Ring signatures differ from group signatures in two fundamental aspects. First, they provide unconditional signer anonymity—there is no mechanism to trace or revoke the anonymity of a specific signature. Second, ring signatures require no centralized setup; any arbitrary set of users can be dynamically selected to form a signing ring without prior coordination or system initialization.

A.1.1 Syntax

A group signature scheme consists of five polynomial-time (PPT) algorithms:

- $\mathsf{Setup}(1^\kappa) \to (\mathsf{gpk}, \mathsf{msk})$: Generates group public key gpk and manager secret key msk.
- $\mathsf{Join}(\mathsf{gpk}, \mathsf{msk}, i) \to \mathsf{sk}_i$: Issues secret signing key sk_i for member i.

- $\mathsf{Sign}(\mathsf{gpk}, \mathsf{sk}_i, m) \to \sigma$: Produces signature σ on message m using member i's key.
- $\mathsf{Verify}(\mathsf{gpk}, m, \sigma) \to \{0, 1\}$: Outputs 1 if σ is valid for m, else 0.
- $\mathsf{Open}(\mathsf{gpk}, \mathsf{msk}, \sigma) \to i$: Traces signer identity i from σ.

A.1.2 Properties

1. **Anonymity**: Given a message and its signature, the identity of the individual signer cannot be determined without the group manager's secret key. The scheme is anonymous if no PPT adversary $\mathcal{A}$ can link signatures to signers. Formally, we define the following security game

Security game $\mathsf{Anon}_{\mathcal{A}}(\kappa)$**:**

$(\mathsf{gpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\kappa)$
$\mathcal{A}(1^\kappa, \mathsf{gpk})$
while $\mathcal{A}$ queries oracles **do**
    Serve Join, Sign, or Open requests
end while
$(m, i_0, i_1) \leftarrow \mathcal{A}$
$b \xleftarrow{\$} \{0, 1\}$
$\sigma \leftarrow \mathsf{Sign}(\mathsf{gpk}, \mathsf{sk}_{i_b}, m)$
$b' \leftarrow \mathcal{A}(\sigma)$
return 1 if $b' = b$, else 0

The scheme satisfies anonymity if for all PPT $\mathcal{A}$:

$$\mathsf{Adv}^{\mathsf{Anon}}_{\mathcal{A}}(\kappa) = \left| \Pr[\mathsf{Anon}_{\mathcal{A}}(\kappa) = 1] - \frac{1}{2} \right| \leq \mathsf{negl}(\kappa).$$

2. **Traceability**: Given any valid signature, the group manager should be able to trace which user issued the signature. This can be formalised by the following security game.

Security game $\mathsf{Trace}_{\mathcal{A}}(\kappa)$**:**

$(\mathsf{gpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\kappa)$
Initialize corruption set $\mathcal{C} \leftarrow \emptyset$
$\mathcal{A}(1^\kappa, \mathsf{gpk})$
while $\mathcal{A}$ interacts **do**
    if Join query for i **then**
        $\mathsf{sk}_i \leftarrow \mathsf{Join}(\mathsf{gpk}, \mathsf{msk}, i)$
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{i\}$
    end if
    Serve Sign queries

10: **end while**
11: $(m^*, \sigma^*) \leftarrow \mathcal{A}$
12: $i^* \leftarrow \mathsf{Open}(\mathsf{gpk}, \mathsf{msk}, \sigma^*)$
13: **return** 1 if:

- $\mathsf{Verify}(\mathsf{gpk}, m^*, \sigma^*) = 1$
- $i^* \notin \mathcal{C}$

The scheme is traceable if $\Pr[\mathsf{Trace}_{\mathcal{A}}(\kappa) = 1] \leq \mathsf{negl}(\kappa)$.

3. **Non-Frameability**: The scheme should be non-frameable, meaning that even if all other group members (and the managers) collude, they cannot forge a signature for a non-participating group member. This is formalized by the following security game. The adversary $\mathcal{A}$ is allowed to issue Join queries for users of its choice, and it can also issue Sign queries for messages of its choice. The adversary is not allowed to issue Open queries for signatures it has created. The adversary wins if it outputs a valid signature σ^* on a message m^* and the group manager is able to open σ^* and identify the signer i^*. The adversary wins if i^* is not in the set of users that it issued Join queries for and i^* did not sign m^*.

Security game $\mathsf{Frame}_{\mathcal{A}}(\kappa)$**:**
1: $(\mathsf{gpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\kappa)$
2: Initialize honest set $\mathcal{H} \leftarrow \emptyset$
3: $\mathcal{A}(1^\kappa, \mathsf{gpk}, \mathsf{msk})$
4: **while** $\mathcal{A}$ interacts **do**
5: **if** Join query for i **then**
6: $\mathsf{sk}_i \leftarrow \mathsf{Join}(\mathsf{gpk}, \mathsf{msk}, i)$
7: $\mathcal{H} \leftarrow \mathcal{H} \cup \{i\}$
8: **end if**
9: Serve Sign queries
10: **end while**
11: $(m^*, \sigma^*) \leftarrow \mathcal{A}$
12: $i^* \leftarrow \mathsf{Open}(\mathsf{gpk}, \mathsf{msk}, \sigma^*)$
13: **return** 1 if:

- $\mathsf{Verify}(\mathsf{gpk}, m^*, \sigma^*) = 1$
- $i^* \in \mathcal{H}$ and i^* didn't sign m^*

The scheme is non-frameable if $\Pr[\mathsf{Frame}_{\mathcal{A}}(\kappa) = 1] \leq \mathsf{negl}(\kappa)$.

A.2 Threshold Signatures

In a (t, n)-threshold signature scheme, any subset of t out of n participants can collaboratively generate a valid signature, while no coalition of fewer than t parties can forge signatures. Unlike group signatures, threshold schemes provide *collective authorization* without any tracing capability.

A.2.1 Syntax

A (t, n)-threshold signature scheme comprises four PPT algorithms:

- $\mathsf{KeyGen}(1^\kappa, t, n) \to (\mathsf{pk}, \{\mathsf{sk}_i\}_{i=1}^n)$: Generates public key pk and secret shares sk_i for n parties.
- $\mathsf{SignShare}(\mathsf{sk}_i, m) \to \sigma_i$: Produces partial signature σ_i from share sk_i on message m.
- $\mathsf{Combine}(\{\sigma_i\}_{i \in S}, m) \to \sigma$: Combines t valid partial signatures into final signature σ, where $|S| \geq t$.
- $\mathsf{Verify}(\mathsf{pk}, m, \sigma) \to \{0, 1\}$: Validates σ against m under pk.

A.2.2 Properties

1. **Unforgeability**: The scheme is unforgeable if no PPT adversary can produce a valid signature without having access to at least t shares. The advantage of an adversary $\mathcal{A}$ in breaking unforgeability is defined as:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{UF}}(\kappa) = \left| \Pr[\mathsf{TS\text{-}UF}_{\mathcal{A}}(\kappa) = 1] - \frac{1}{2} \right| \leq \mathsf{negl}(\kappa).$$

   The unforgeability game is defined as follows:

Security game $\mathsf{TS\text{-}UF}_{\mathcal{A}}(\kappa)$:

$(\mathsf{pk}, \{\mathsf{sk}_i\}_{i=1}^n) \leftarrow \mathsf{KeyGen}(1^\kappa, t, n)$
$\mathcal{C} \leftarrow \mathcal{A}(\mathsf{pk})$
Initialize $\mathcal{Q} \leftarrow \emptyset$
while $\mathcal{A}$ adaptively queries $\mathsf{SignShare}(m_j)$ **do**
    $\sigma_i \leftarrow \mathsf{SignShare}(\mathsf{sk}_i, m_j)$ for $i \notin \mathcal{C}$
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m_j\}$
end while
$(m^*, \sigma^*) \leftarrow \mathcal{A}$
return $[[\mathsf{Verify}(\mathsf{pk}, m^*, \sigma^*) = 1 \wedge m^* \notin \mathcal{Q}]]$

2. **Robustness**: The scheme is robust if it guarantees that any set of t honest parties can produce a valid signature. This means that even if some parties are corrupted,

the remaining honest parties can still generate a valid signature. The robustness game is defined as follows:

1: $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i=1}^n) \leftarrow \mathsf{KeyGen}(1^\kappa, t, n)$
2: $S \subseteq [n]$ with $|S| \geq t$
3: $\{\sigma_i \leftarrow \mathsf{SignShare}(\mathsf{sk}_i, m)\}_{i \in S}$
4: $\sigma \leftarrow \mathsf{Combine}(\{\sigma_i\}_{i \in S}, m)$
5: **return** 1 if $\mathsf{Verify}(\mathsf{pk}, m, \sigma) = 0$

Scheme is robust if $\Pr[\text{Output } 1] = 0$.

Throughout, we assume a standard honest-majority model: typically one takes $n \geq 2t+1$, so that at most $t-1$ parties can be corrupted. We also assume that all public keys pk_i are certified (via a PKI) and that communication is private and authenticated between parties. These assumptions ensure that the adversary cannot forge signatures or modify honest parties' communications.

In this chapter, we have discussed two variants of signature schemes: group signatures and threshold signatures. Group signatures allow a group of users to sign messages on behalf of the group, while threshold signatures enable a subset of participants to collaboratively generate a valid signature. Both schemes are closely related to ring signatures, but they have different properties and use cases. A natural question to ask is whether we can use the framework defined in this thesis to construct and generalize these schemes.