UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Mihkel Jaas Karu

Design and Security Analysis of Blind Smart-ID/SplitKey Signature

Bachelor's Thesis (9 ECTS)

Supervisor(s): Nikita Snektov, MSc
Peeter Laud, PhD
Toomas Krips, PhD

Tartu 2025

Design and Security Analysis of Blind Smart-ID/SplitKey Signature

Abstract:
A blind signature scheme is a cryptographic primitive that enables a user to obtain a signature on a message without revealing its content to the signer. This unique property ensures both the authenticity of the signature and the privacy of the message, making blind signatures particularly useful in applications requiring anonymity, such as electronic voting and digital cash systems.
SplitKey [BKLO17] is a cryptographic technology designed to enable the creation of digital signatures using a private key distributed across two devices. However, it currently lacks support for blind signing.
The goal of this thesis is to determine whether the original SplitKey design can be modified to create a blind signature scheme, and to explore how it could be done. To achieve this, the thesis presents a novel blind signature scheme, RSA-BSK, which extends the functionality of the SplitKey signature scheme to support blind signatures. We provide a detailed security analysis of RSA-BSK, including formal proofs of correctness, unforgeability, and blindness, thus establishing its suitability for privacy-preserving applications in a distributed key setting.

Keywords:
Smart-ID, blind signature scheme, threshold cryptography, Multiparty RSA, SplitKey, Server-Supported RSA.

CERCS: P170 Computer science, numerical analysis, systems, control

Pimeallkirjastamise Smart-ID/SplitKey Skeemi Disain ja Turbeanalüüs

Lühikokkuvõte:
Pimeallkirjastamisskeem on krüptograafiline primitiiv, mis võimaldab kasutajal panna allkirja sõnumile, ilma et sõnumi sisu oleks allkirjastajale nähtav. See omadus tagab nii allkirja autentsuse kui ka sõnumi privaatsuse, muutes pimeallkirjad oluliseks näiteks elektroonilises hääletamises ja e-raha süsteemides, kus anonüümsus on hädavajalik.
SplitKey on krüptograafiline tehnoloogia [BKLO17], mis võimaldab luua digiallkirju, kasutades kahe seadme vahel jagatud privaatvõtit. Hetkel puudub SplitKey-l pimeallkirjastamise võimekus.
Selle lõputöö eesmärk on välja selgitada, kas SplitKey algset disaini saab muuta nii, et see toetaks pimeallkirjastamist, ning uurida, kuidas seda oleks võimalik teostada. Eesmärgi saavutamiseks esitab käesolev lõputöö uue pimeallkirjastamisskeemi, RSA-

BSK, mis laiendab SplitKey allkirjastamisskeemi funktsionaalsust, lisades toe pimeallkirjastamiseks. Viime läbi põhjaliku turbeanalüüsi, esitades formaalsed tõendid võltsimiskindluse ja pimeduse kohta, tõestades sellega RSA-BSK sobivust privaatsust nõudvatesse rakendustesse hajutatud võtmehalduskeskkonnas.

Võtmesõnad:
Smart-ID, pimesiknatuuriskeem, lävend krüptograafia, mitmepoolne RSA, SplitKey, Server-toega RSA.

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Contents

1 Introduction

Cryptography is the cornerstone of modern information security, enabling secure communication, integrity and authenticity of exchanged data in an increasingly digital world.

The authenticity of a message can be ensured using digital signature schemes. In such a scheme, the signer generates a signature for a given message using a private key known only to them. Once the signature is created, anyone can verify the message's authenticity by checking that the signature corresponds to the message and was produced with the associated private key. The verification process does not use the private key but instead relies on a public key that is cryptographically linked to the private key. This key pair is generated in advance, before any messages are signed.

Blind signature schemes allow a signer to produce a valid digital signature on a message chosen by a second party, without being able to learn the content of the message. This is achieved by first *blinding* the message, which involves obscuring it using cryptographic techniques, before sending it to the signer. The signer then produces a signature on the blinded data, and the recipient later *unblinds* it to obtain a valid signature on the original message. The primary motivation behind blind signatures is privacy preservation. Blind signatures provide an elegant solution in scenarios where authentication and non-repudiation are required, but where disclosing the message itself would be undesirable. Blind signature schemes play an important role in electronic payment systems[1], anonymous voting [Wil23], and private transactions [WOW08].

Smart-ID[2] is a widely adopted digital identity solution in the Baltics, Belgium and Iceland, enabling citizens to authenticate themselves, sign documents, authorize transactions, and access e-services securely over the internet. At its core, Smart-ID employs the SplitKey[3] cryptographic algorithm, which facilitates secure key management and digital signing operations. SplitKey is a threshold cryptography technology that allows creating digital signatures using a private key that is split across two devices. This prevents either party from using the private key without the cooperation of the other party, making it infeasible to produce a signature without compromising both devices.

1.1 Related work

Several threshold blind signature schemes have been proposed to combine the benefits of threshold cryptography and blind signatures, aiming to enhance both robustness and privacy [VZK03, KM14, CKM+23, KRB+24, LNÖ25]. These schemes typically enable a set of signers to collaboratively issue a blind signature without any single signer having full signing authority, thus improving resistance to key compromise or coercion. However, existing approaches fall short of achieving our desired goals in key aspects. Although

[1] https://www.taler.net/en/

[2] https://www.smart-id.com/

[3] https://cyber.ee/solutions/digital-identity/#splitkey-unique-technology

threshold signing is similar to SplitKey signing [BKLO17], the key splitting property of SplitKey is hard to replicate without changing the threshold signature scheme. These limitations motivate the need for a new protocol that could provide this key splitting property.

1.2 Contribution

The goal of this thesis is to determine whether a blind signature scheme can be designed using SplitKey, and to explore how it could be done. To meet that goal, we design a blind signature scheme based on the existing SplitKey algorithm and conduct a security analysis on it. To achieve this, we propose a new protocol called RSA-BSK (Blind SplitKey), which integrates blind signing capabilities into the existing SplitKey framework. The design of RSA-BSK builds upon "Security Analysis of RSA-BSSA" [Lys22], which uses the RSA-PSS signature scheme [BR96] as a basis to design a blind signature scheme RSA-BSSA. In the security analysis we will provide four key security proofs: correctness of the composition procedure, blindness, strong one-more unforgeability against a malicious server, and strong one-more unforgeability against a malicious client. These proofs will primarily rely on reductions from the already proven secure RSA-BSSA scheme [Lys22], demonstrating that the security properties are preserved with only negligible differences. In this thesis RSA-BSK is proven to be at least as secure as RSA-BSSA.

1.3 Structure of the thesis

The structure of this work is as follows: Section 2 provides the necessary background, including notation, mathematical preliminaries, and security definitions, to help the reader understand the thesis. Section 3 presents background on the cryptographic algorithms necessary to understand the RSA-BSK protocol. Section 4 provides an overview of blind signature schemes, including detailed descriptions of blind RSA and RSA-BSSA. Section 5 introduces the proposed RSA-BSK protocol, outlining its design and operation. Section 6 offers a security analysis of RSA-BSK. Section 7 outlines potential directions for future work, followed by conclusions in the final section.

2 Preliminaries

2.1 Notation

- The symbol $||$ denotes the concatenation of two inputs.
- The symbol $\oplus$ is exclusive disjunction operation (XOR).
- $a \leftarrow b$ is assigning the value of the variable a to the value b.
- $a \xleftarrow{\$} A$ is sampling an element a uniformly at random from the set A.
- The symbol $\perp$ is used to indicate a failure or abort.
- $\{0,1\}^l$ is a string of length $l \in \mathbb{N}$ that consists of 0 and 1 (called a l long bit-string).

2.2 Mathematical preliminaries

1. For $x, n \in \mathbb{N}$, we say that n divides x (or x is a multiple of n), and write $n \mid x$, if there exists an integer k such that $x = k \cdot n$.
2. For $n \in \mathbb{N}$, we say that integers a and b are congruent modulo n, and write $a \equiv_n b$ or $a \equiv b \ (mod\ n)$, if $n|(a-b)$.
3. The greatest common divisor for two numbers $x, n \in \mathbb{N}$ is the largest number $k \in \mathbb{Z}$ that satisfies the equations, where $a, b \in \mathbb{Z}$: $\begin{cases} x = k \cdot a \\ n = k \cdot b \end{cases}$, we notate it as $\gcd(x, n) = k$.
4. Euler totient function (denoted as $\phi(N)$) takes in as input the number N and returns the size of the set $\{x \in \mathbb{Z}_N \mid \gcd(N, x) = 1\}$. In other words, the number of integers in range 1 and $N-1$ (inclusive) that are co-prime to N.
   For example: $\phi(7) = 6$ (size of $\{1,2,3,4,5,6\}$), $\phi(8) = 4$ (size of $\{1,3,5,7\}$).
5. The group of elements $\mathbb{Z}_N^*$ represents the set of integers between 1 and $N-1$ that are co-prime to N: $\mathbb{Z}_N^* = \{a \in \mathbb{Z} \mid 1 \leq a < N \ \wedge \ \gcd(a, N) = 1\}$.
6. The modular inverse of an integer $a \in \mathbb{Z}_N^*$ is an integer $x \in \mathbb{Z}_N^*$ (denoted as a^{-1}) such that $a \cdot x \equiv 1 \pmod{N}$.
7. Division under modulo works as such: For $a, b, c, n \in \mathbb{Z}$, $(a/b = c) \wedge (\gcd(a, n) = 1) \implies a \cdot b^{-1} \equiv c \pmod{n}$.
   For example: $90/9 = 10 \implies 90 \cdot 13 \equiv 10 \pmod{29}$ because $9 \cdot 13 \equiv 1 \pmod{29}$.

8. According to the Euler-Fermat theorem, if $n \in \mathbb{Z}_N^*$, then $n^{\phi(N)} \equiv 1 \pmod{N}$.

9. A prime number p is an (ℓ, s)-safe prime, if $p = 2 \cdot a \cdot p'_1 \cdots p'_k + 1$, where $p'_i > s$ are prime numbers, and $1 \leq a \leq \ell$.

10. Chinese Remainder Theorem (later referred to as CRT) states that if $n_1, n_2, \ldots, n_k$ are pairwise relatively prime, then the system of congruences

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\cdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

has a unique solution modulo $n = n_1 \cdot n_2 \cdots n_k$.

2.3 Security definitions

Definition 1 (Negligible advantage)**.** *An adversary's advantage is said to be negligible if it is smaller than the inverse of any polynomial in the security parameter λ. For practical purposes, in the case of 128-bit security, any advantage $\delta \ll \frac{1}{2^{128}}$ is said to be negligible.*

Definition 2 (Distributed signature protocol [SVL24a])**.** *A distributed signature protocol between parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ consists of the following algorithms:*

- $\mathsf{Setup}(1^\lambda)$ *is an algorithm that takes as input a security parameter λ and generates a set of public parameters* par *for the protocol.*
- $\mathsf{KeyGen}_j(\mathsf{par})$ *is an interactive key generation protocol that is run by each party P_j, which takes as input public parameters* par *and outputs a public key* pk *and the secret key share sk_j of P_j.*
- $\mathsf{Sign}_j(\mathsf{sk}_j, m)$ *is an interactive signing protocol that is run by each party P_j. It takes as input a secret key share sk_j of a party P_j, and a message m from the Client and outputs a single signature σ to the Client.*
- $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$ *is a verification algorithm that takes as input a public key* pk*, a message m, and a signature σ. It outputs 1 if the signature is valid and 0 if it is invalid.*

Definition 3 (Correctness)**.** *A distributed signature protocol is correct if for any message $m \in \{0,1\}^*$ it holds that:*

$$Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{pk}, m, \sigma) = 1 : \\ \mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda); \\ \{\mathsf{sk}_j\}_{j=1}^n \leftarrow \mathsf{KeyGen}(\mathsf{par}); \\ \sigma \leftarrow \{\mathsf{Sign}_j(\mathsf{sk}_j, m)\}_{j=1}^n \end{array}\right] = 1$$

Definition 4 (Blindness)**.** *We have a challenger E and an adversary $\mathcal{A}$ that play a game* $\mathsf{Game}^{blind}_{E,\mathcal{A}}(\lambda)$*. Let the advantage of an adversary $\mathcal{A}$ against the blindness game* $\mathsf{Game}^{blind}_{E,\mathcal{A}}(\lambda)$ *be:*

$$\mathsf{Adv}^{blind}_{E,\mathcal{A}}(\lambda) = |Pr[\mathsf{Game}^{blind}_{E,\mathcal{A}}(\lambda) = 1] - 1/2|$$

A distributed signature protocol satisfies ***blindness*** *if for all PPT adversaries $\mathcal{A}$,* $\mathsf{Adv}^{blind}_{E,\mathcal{A}}(\lambda)$ *is negligible.*

MAIN $\mathsf{Game}^{\text{blind}}_{E,\mathcal{A}}(\lambda)$

$\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda)$

$(\mathsf{msg}_0, \mathsf{msg}_1) \leftarrow \mathcal{A}^{\mathsf{msg} \in \{0,1\}^*}$

$b \xleftarrow{\$} \{0,1\}$

$(\sigma_b, \sigma_{\{1-b\}}) \leftarrow (\{\mathsf{Sign}_j(\mathsf{sk}_j, \mathsf{msg}_b)\}_{j=1}^n, \{\mathsf{Sign}_j(\mathsf{sk}_j, \mathsf{msg}_{\{1-b\}})\}_{j=1}^n)$

if $\mathsf{VerifyPSS}(\mathsf{pk}, \mathsf{msg}_0, \sigma_0) = \mathsf{VerifyPSS}(\mathsf{pk}, \mathsf{msg}_1, \sigma_1) = 1$:

  $(\sigma_0, \sigma_1) \rightarrow \mathcal{A}$

else : $\perp \rightarrow \mathcal{A}$

$b' \leftarrow \mathcal{A}^{\text{guess from } \{0,1\}}$

return 0 **if** $b' \neq b$

return 1

Figure 1. The blindness game for a distributed signature protocol.

Definition 5 (Perfect blindness)**.** *A distributed signature protocol satisfies* ***perfect blindness*** *if for all PPT adversaries $\mathcal{A}$,* $\mathsf{Adv}^{blind}_{E,\mathcal{A}}(\lambda) = 0$*.*

Definition 6 (Strong One-More Unforgeability)**.** *Let the advantage of an adversary $\mathcal{A}$ against the strong one-more unforgeability game against a malicious client* $\mathsf{Game}^{somuf}_{E,\mathcal{A}}(\lambda)$*, be as follows:*

$$\mathsf{Adv}^{somuf}_{E,\mathcal{A}}(\lambda) = |Pr[\mathsf{Game}^{somuf}_{E,\mathcal{A}}(\lambda) = 1]|.$$

A distributed signature protocol satisfies strong one-more unforgeability against a malicious client if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{somuf\text{-}C}_{E,\mathcal{A}}(\lambda)$ *is negligible. Game is described in Section 6.3.*

Definition 7 (Strong One-More Unforgeability against malicious server)**.** *Let the advantage of an adversary $\mathcal{A}$ against the strong one-more unforgeability against malicious server game* $\mathsf{Game}^{somuf\text{-}S}_{E,\mathcal{A}}(\lambda)$*, be as follows:*

$$\mathsf{Adv}^{\textit{somuf-S}}_{E,\mathcal{A}}(\lambda) = |\textit{Pr}[\mathsf{Game}^{\textit{somuf-S}}_{E,\mathcal{A}}(\lambda) = 1]|.$$

A distributed signature protocol satisfies strong one-more unforgeability against malicious server if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{\textit{somuf-S}}_{E,\mathcal{A}}(\lambda)$ is negligible. Game is described in Section 6.4.

Definition 8 (Strong One-More Unforgeability against malicious client)**.** *Let the advantage of an adversary $\mathcal{A}$ against the strong one-more unforgeability game against a malicious client $\mathsf{Game}^{\textit{somuf-C}}_{E,\mathcal{A}}(\lambda)$, be as follows:*

$$\mathsf{Adv}^{\textit{somuf-C}}_{E,\mathcal{A}}(\lambda) = |\textit{Pr}[\mathsf{Game}^{\textit{somuf-C}}_{E,\mathcal{A}}(\lambda) = 1]|.$$

A distributed signature protocol satisfies strong one-more unforgeability against a malicious client if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{\textit{somuf-C}}_{E,\mathcal{A}}(\lambda)$ is negligible. Game is described in Section 6.5.

3 Signature schemes

3.1 Multi-prime RSA signature scheme

To get a better understanding of how a RSA based signature scheme functions we will be describing multi-prime RSA as is described by M. Jason Hinek [Hin08]:

For any integer $r \geq 2$, r-prime RSA signature scheme consists of the following three algorithms:

Key Generation: Let N be the product of r randomly chosen distinct primes $p_1, ..., p_r$. Compute Euler's totient function of N: $\phi(N) = \prod_{i=1}^{r}(p_i - 1)$. Choose an integer e, $1 < e < \phi(N)$, such that $\gcd(e, \phi(N)) = 1$. The pair (N, e) is the public key. Compute the unique $d \in \mathbb{Z}^*_{\phi(N)}$ such that $e \cdot d \equiv 1 \pmod{\phi(N)}$. The private key is the pair (N, d).

Signing: For any message $m \in \mathbb{Z}^*_N$, the signature is computed as $s = m^d \pmod{N}$.

Verification: For any signature $s \in \mathbb{Z}^*_N$, the message is recovered by computing $m = s^e \pmod{N}$.

When $r = 2$ we have the original RSA signature scheme. For all $r > 2$ we call it r-prime RSA.

3.2 Multiparty RSA signature scheme

Multiparty RSA signature scheme employs similar mathematical procedures to RSA, but relies on the fact that the private exponent d is distributed between many signing parties. This is done to ensure that no coalition of parties, except the coalition of all of them could sign an entire message.

Damgard-Mikkelsen-Skeltved Scheme [DMS15] is described as such:

Key Generation

1. Two parties P_1 and P_2 agree on a public exponent e. Then they generate a standard RSA key pair each: $\mathsf{pk}_{P_1} = (N_{P_1}, e), \mathsf{sk}_{P_1} = (N_{P_1}, d_{P_1})$ and $\mathsf{pk}_{P_2} = (N_{P_2}, e), \mathsf{sk}_{P_2} = (N_{P_2}, d_{P_2})$ respectively.
2. Parties exchange public keys and generate joint public key $N = N_{P_1} \cdot N_{P_2}$.

Distributed Decryption/Signing

1. On input $x \in \mathbb{Z}_N^*$, P_1 and P_2 compute, for $i \in \{1,2\}$, $s_{P_i} = x^{d_{P_i}} \pmod{N_{P_i}}$ and exchange these values.

2. Both parties use CRT to compute $s \in \mathbb{Z}_N^*$ such that for $i \in \{1,2\}$, $s \equiv s_{P_i} \pmod{N_{P_i}}$. They check that $s^e \equiv x \pmod{N}$ and output s if this is the case. Otherwise, abort.

Verification: For any signature $s \in \mathbb{Z}_N^*$, the message is recovered by computing $m = s^e \pmod{N}$.

3.3 RSA-PSS

We describe RSA-PSS as it was established in "The Exact Security of Digital Signatures - How to Sign with RSA and Rabin" [BR96].

Signature scheme $\mathsf{PSS}[k_0, k_1]$ = (genPSS, SignPSS, VerifyPSS) is parameterized by k_0 and k_1, which are numbers between 1 and k, satisfying $k_0 + k_1 \leq k - 1$.

The key generation algorithm genPSS is identical to standard RSA KeyGen: run Setup(1^k) to obtain (N, e, d), and output $(\mathsf{pk}, \mathsf{sk})$, where $\mathsf{pk} = (N, e)$ and $\mathsf{sk} = (N, d)$.

The signing and verifying algorithms make use of two hash functions. The first, h, called the compressor, maps as $h : \{0,1\}^* \to \{0,1\}^{k_1}$ and the second, g, called the generator, maps as $g : \{0,1\}^{k_1} \to \{0,1\}^{k-k_1-1}$. Let g_1 be the function which on input $w \in \{0,1\}^{k_1}$ returns the first k_0 bits of $g(w)$, and let g_2 be the function which on input $w \in \{0,1\}^{k_1}$ returns the remaining $k - k_0 - k_1 - 1$ bits of $g(w)$.

We now describe how the SignPSS and VerifyPSS functions work:

Algorithm 1: SignPSS(M)

$r \xleftarrow{\$} \{0,1\}^{k_0}$; $w \leftarrow h(M \,||\, r)$; $r^* \leftarrow g_1(w) \oplus r$

$x \leftarrow 0 \,||\, w \,||\, r^* \,||\, g_2(w)$

return $x^d \pmod{N}$;

Algorithm 2: VerifyPSS(M, y)

$x \leftarrow y^e \pmod{N}$

Break up x as $b \,||\, w \,||\, r^* \,||\, \gamma$

$r \leftarrow r^* \oplus g_1(w)$

if $(h(M \,||\, r) = w \;\;\&\;\; g_2(w) = \gamma \;\;\&\;\; b = 0)$ **then return** 1

return $\perp$

The value x is representative of adding **PSS encoding** on a message M. A value x is a correct PSS encoding of M if:

1. The first bit of x represented as b is equal to 0.
2. The next k_1 bits represented as w is equal to $h(M \mid\mid r)$, where $r \xleftarrow{\$} \{0,1\}^{k_0}$.
3. The next k_0 bits represented as r^* is equal to $g_1(w) \oplus r$.
4. The remaining bits represented as γ is equal to $g_2(w)$.

This will be important to understand how VerifyPSS works later on in the thesis.

Applying PSS encoding on a message, that is sent to signing, allows the message to be longer then normally. This is the case because a longer message is reduced in size after being hashed.

3.4 SplitKey RSA

SplitKey [BKLO17] uses the functionalities of Multiparty-RSA (Distributed Multiprime RSA) [DMS15], Shared RSA [Des88, DF90], and RSA-PSS [BR96], while adding some security elements to construct a signature scheme between a server and mobile device. For a better understanding we have provided a description of the protocol below, in Figures 2, 3 and their visual representation in Figures 8, 9.

This will not be an exact protocol that has been implemented in Smart-ID, but a close representation. We have presented the User as someone using their device (the Client) to communicate with the Server. Different clients use different functions Φ in the genShare function. In practice, $\Phi(\cdot)$ is replaced by a pseudo-random function $F(u, \cdot)$, where u is a sufficiently long random bit-string. $\sigma = C_{n_1,n_2}(s_1, s_2)$ means using CRT to compute $\sigma \in \mathbb{Z}_N^*$ such that for $i \in \{1,2\}$, $\sigma \equiv s_i \pmod{n_i}$. T_0 is denoted as the maximum number of wrong password guesses for a client, by default $T_0 = 3$. The value r is used to detect if a unauthorized device has requested a signature with the correct d_1' value.

Algorithm 3: $\mathsf{genShare}^{\Phi}(pwd, n_1)$

```
for s ∈ {0, 1, 2, ..., 255} do
    d'_1 ← Φ(pwd||s);
    if d'_1 < n_1 then
        return d'_1;
return ⊥;
```

Key Generation(λ)

Setup

- From λ, suitable values for ℓ, s, and RSA modulus length k are selected.
- Threshold of attempts T_0, and the public exponent e, where $\gcd(\phi(n), e) = 1$, are fixed.
- All communication between Client and Server take place over secure channels.

Client side

1. Client samples two (ℓ, s)-safe primes p_1, q_1.
2. Client computes $n_1 = p_1 \cdot q_1$, $d_1 \equiv e^{-1} \pmod{\phi(n_1)}$.
3. Client receives password pwd from User, generates and stores random bit-string u and then computes $d_1' = \text{genShare}^{F(u,\cdot)}(pwd, n_1)$ and $d_1'' = d_1 - d_1' \pmod{\phi(n_1)}$.
4. Client sends (d_1'', n_1) to the Server.

Server side

1. Server samples two (ℓ, s)-safe primes p_2, q_2.
2. Server computes and stores $n_2 = p_2 \cdot q_2$, $n = n_1 \cdot n_2$ and $d_2 \equiv e^{-1} \pmod{\phi(n_2)}$.
3. Server generates and stores random bit-string r_s.
4. Server takes $T = T_0$ and stores $(n_1, n_2, d_1'', d_2, r_s, T)$
5. Server sends (n, r) back to the Client, where $r = r_s$.

Client side

1. Client stores (n, n_1, u, r) and securely deletes all other values.

Outcome

- The public key of the Client is (N, n_1, e).
- The public key of the Server is (N, n_2, e).
- The private key of the Client is d_1', that can be computed with correct input pwd to $\text{genShare}^{F(u,\cdot)}$.
- The private key of the Server is (d_1'', d_2).

Figure 2. Key generation for SplitKey

Signing(M)
Client side

1. Client gets password pwd from User and finds $d_1' = \text{genShare}^{F(u,\cdot)}(pwd, n_1)$.
2. Client creates the signature share $y = H(M)^{d_1'} \pmod{n_1}$.
3. Client sends (y, m, r) to the Server.

Server side

1. Server looks up Client's record $(n_1, n_2, d_1'', d_2, r_s, T)$.
2. Server checks if $r_s = r$, if it doesn't match then *deactivate Client*.
3. Server generates a random bit-string r'
4. Server computes Client's signature $s_1 = y \cdot m^{d_1''} \pmod{n_1}$.
5. Server verifies the signature:
   $[\mathsf{VerifyPSS}(s_1, e, m, n_1)]$ { *if 1, correct signature*; if $\perp$, decrement T and [T = 0] { *if 1, deactivate Client.*; *if $\perp$, drop request.* } }
6. Server computes $s_2 = m^{d_2} \pmod{n_2}$ and creates the composite signature $s = C_{n_1,n_2}(s_1, s_2)$.
7. Server sends (s, m, r') to Client and assigns r' to r_s and T_0 to T.

Client side

1. Client receives signature and verifies it.
2. Client replaces its stored r with r'.

Figure 3. Signing algorithm for SplitKey

4 Blind signature schemes

The first cryptographic signature scheme that can be regarded as a blind signature scheme was introduced in a research paper by David Chaum [Cha82]. The paper demonstrates how a bank could exchange money between two clients without knowing who the exchange is between. This paper introduced the concept of blind signatures and presented the first practical construction using the RSA algorithm — effectively creating the first RSA blind signature scheme.

4.1 RSA blind signature scheme

To understand RSA based blind signature schemes more intuitively we will be describing the RSA blind signature scheme [Cha82]. The interactions are described to be between a client and a server.

Key Generation. Let p and q be distinct primes, and let $N = p \cdot q$. N is called the RSA modulus. Let e and d be integers such that $e \cdot d \equiv 1 \pmod{\phi(N)}$. That is, e and d are multiplicative inverses mod $\phi(N)$. We treat (N, e) as the public key and (N, d) as the private key.

Signing protocol.

1. **Blinding.** Client samples r randomly from set $\mathbb{Z}_N^*$ and computes $x = m \cdot r^e \pmod{N}$, where m is the message they wish to be signed and x is the blinded message.
2. **Signing.** Client sends the server the value x. Then the server uses the private key to compute $y = x^d \pmod{N}$. Finally the server sends y back to the client.
3. **Finalize.** Client computes $s = y/r \pmod{N}$.

Verification. To verify signature compute if $m = s^e \pmod{N}$.

This follows almost like the original RSA signature scheme, but with the addition of a blinding and finalizing function.

4.2 RSA-BSSA

RSA-BSSA uses the RSA-PSS [BR96] signature scheme as a basis for a blind signature scheme. For this thesis it is not necessary to understand the functions precisely, since it does not affect our findings. But we will briefly add what some of the functions do:

- $\mathsf{I2OSP}(z, kLen)$: Turns integer z to an $kLen$ long octet-string (byte string).

- $\mathsf{OS2IP}(z)$: Turns an octet-string in to an integer.
- $\mathsf{PSSEncode}(\mathsf{msg}, k)$: Applies an PSS encoding to a string msg.
- $\mathsf{VerifyPSS}(\mathsf{pk}, \mathsf{msg}, \sigma)$: Checks if σ is a valid signature of msg and if a correct PSS encoding has been applied to msg before signing.

For a better understanding see "The Exact Security of Digital Signatures - How to Sign with RSA and Rabin" [BR96].
We will describe it as it is described in "Security Analysis of RSA-BSSA" [Lys22]:

We begin by describing the basic scheme in which the user obtains from the signer an RSA-PSS signature on the message msg. As usual, the scheme consists of a key generation algorithm, a protocol for obtaining signatures, and a signature verification algorithm.

Key generation. The key generation algorithm is the standard RSA key generation: The public key is $\mathsf{pk} = (n, e)$, where n is an RSA modulus of length λ (given as 1^λ), and e is relatively prime to $\phi(n)$. The secret key is $\mathsf{sk} = (n, d)$ such that $e \cdot d \equiv 1 \pmod{\phi(n)}$.

Blind signing protocol. The protocol consists of three algorithms: $\mathsf{Blind}, \mathsf{BSig}$, and $\mathsf{Finalize}$. On a input message msg that the client wishes to get a signature for, the client runs $\mathsf{Blind}(\mathsf{pk}, \mathsf{msg})$ and obtains (blinded_msg, inv). The server runs the algorithm BSig(pk, blinded_sig, inv) to derive the signature σ. The three algorithms are as follows:

$\mathsf{Blind}(\mathsf{pk}, m)$: Compute a PSS encoding of msg : $\mathsf{EM} = \mathsf{PSSEncode}(\mathsf{msg}, k-1)$, and let $m = \mathsf{OS2IP}(\mathsf{EM})$ be the corresponding integer. Next, sample $r \xleftarrow{\$} \mathbb{Z}_n^*$, compute $z = m \cdot r^e \pmod{n}$, and make sure that $z \in \mathbb{Z}_n^*$. Compute $r_{inv} = r^{-1} \pmod{n}$, and output (z, r_{inv}) as octet strings i.e. byte strings, output blinded_msg $= \mathsf{I2OSP}(z, kLen)$, inv $= \mathsf{I2OSP}(r_{inv}, kLen)$.

BSig(sk, blinded_msg): First, check that the string blinded_msg is of bit length k, and reject if it is not. Next, convert into a k-bit integer $m = \mathsf{OS2IP}$(blinded_msg). Output the binary representation of $s = m^d \pmod{n}$, i.e. blinded_msg $= \mathsf{I2OSP}(s, kLen)$.

$\mathsf{Finalize}$(pk, msg, blinded_sig, inv) Convert blinded_sig and inv into integers using the $\mathsf{OS2IP}$ procedure: $z = \mathsf{OS2IP}$(blinded_sig), $r_{inv} = \mathsf{OS2IP}$(inv); compute $s = z \cdot r_{inv} \pmod{n}$. The signature is the binary representation of s i.e. $\sigma = \mathsf{I2OSP}(s, kLen)$. Finally, if $\mathsf{VerifyPSS}(\mathsf{pk}, msg, \sigma)$, then output σ, else fail.

Verification. The verification algorithm calls $\mathsf{VerifyPSS}(\mathsf{pk}, msg, \sigma)$.

5 RSA-BSK signature scheme

In this thesis we will take the earlier described cryptographic algorithm SplitKey [BKLO17, SVL24b] and convert it in to a blind signature scheme, named RSA-BSK.

The RSA-BSK protocol consists of three processes – Key generation, Signing, Verification:

Key Generation: Since adding blinding does not change the key generation process, RSA-BSK uses the same key generation process as SplitKey, described in Figure 2 and visualized in Figure 8.

Signing: The signing process, described in Figure 4, has some extra steps added on the client side (**bold in** Figure 10), like the blinding function Blind and different verification function Verify.

Algorithm 4: $\mathsf{Blind}(m, e, N)$

while $True$:
    $R \xleftarrow{\$} \mathbb{Z}_N^*$;
    if $gcd(R, N) = 1 : break$;
$m_* \leftarrow m \cdot R^e \pmod N$;
return (m_*, R);

Algorithm 5: $\mathsf{Verify}(s, e, m, n)$

if $s^e = m \pmod n$;
    return 1;
return $\perp$;

Signing(msg)
Client side

1. Client gets password pwd from User and finds $d_1' = \text{genShare}^{F(u,\cdot)}(pwd, n_1)$.
2. Client computes PSS encoding of message: $M = H(\text{msg})$.
3. Client runs the function $\text{Blind}(M, e, N)$ and receives (m_*, R)
4. Client creates the signature share $y = m_*^{d_1'} \pmod{n_1}$.
5. Client sends (y, m_*, r) to the Server.

Server side

1. Server looks up Client's record $(n_1, n_2, d_1'', d_2, r_s, T)$.
2. Server checks if $r_s = r$, if it doesn't match then *deactivate Client*.
3. Server generates a random bit-string r'
4. Server computes Client's signature $s_1 = y \cdot m^{d_1''} \pmod{n_1}$.
5. Server verifies the signature:

   $[\text{Verify}(s_1, e, m, n_1)]$ { *if 1, correct signature*
   if $\perp$, decrement T and [T = 0] { *if 1, deactivate Client.* / *if $\perp$, drop request.* }
6. Server computes $s_2 = m^{d_2} \pmod{n_2}$ and creates the composite signature $s_* = C_{n_1,n_2}(s_1, s_2)$.
7. Server sends (s_*, m, r') to Client and assigns r' to r_s and T_0 to T.

Client side

1. Client computes the signature $\sigma = s_*/R \pmod{N}$.
2. Client receives signature and verifies it.
3. Client replaces its stored r with r'.

Figure 4. Signing algorithm for RSA-BSK

Verification: Uses VerifyPSS as its verification function, described in Algorithm 2.

It is essential to employ two distinct verification functions – Verify and VerifyPSS – for validating σ and s_1, respectively. This distinction arises from the use of PSS encoding within the protocol. The final signature σ adheres to the PSS format and must therefore be verified using the VerifyPSS function. In contrast, s_1 is produced over a blinded message that does not conform to PSS encoding, even if the underlying message itself may be PSS-encoded. As a result, s_1 must be verified using a function that does not enforce PSS-specific checks, such as Verify.

6 Security analysis of RSA-BSK

A blind signature scheme is secure if it holds correctness, blindness and unforgeability. We are choosing to prove unforgeability with strong one-more unforgeability, since we will be relating the unforgeability of RSA-BSK to RSA-BSSA, which is proven strong one-more unforgeable [Lys22].

Since we are only changing the signing part of the SplitKey [BKLO17] protocol then we will not prove any part of the Key Generation procedure. For that we have the proof of the underlying protocol SplitKey [BKLO17, SVL24b].

6.1 Correctness of the composition procedure under blindness

Let us prove the correctness of the composition procedure when blindness is introduced in the signature:
Let us assume that $s_1 = m^{d_1} \pmod{n_1}$ and $s_2 = m^{d_2} \pmod{n_2}$ are two signatures on a message m. The composition procedure uses CRT to compute $\sigma \in \mathbb{Z}_N^*$ such that for $i \in \{1, 2\}$, $\sigma \equiv s_i \pmod{n_i}$, when $N = n_1 \cdot n_2$ and $\gcd(n_1, n_2) = 1$.

From Gauss's algorithm [MOV97] we know that we can represent σ in the form of an equation $\sigma = \sum_{i=1}^{2} s_i \cdot N_i \cdot y_i \pmod{N}$, where $N_i = N/n_i$ and $y_i = N_i^{-1} \pmod{n_i}$.

From this we can derive that if $s_* = \sigma \cdot r \equiv \sum_{i=1}^{2} s_i \cdot r \cdot N_i \cdot y_i \pmod{N}$ where $r \in \mathbb{Z}_N^*$, then $\sigma = s_* \cdot r^{-1} \pmod{N}$.

6.2 Blindness of the blind signature scheme

6.2.1 Preliminaries

To show the blindness of the RSA-BSK blind signature scheme we will be using Lemmas proven in "Security analysis of RSA-BSSA" [Lys22]:

Lemma 1. *Let $N > 1$ be any integer, let $N = \prod_{i=1}^{\ell} p_i^{\alpha_i}$ be its prime factorization. Then $\mathbb{Z}_N^*$ is of size $\phi(N) = \prod_{i=1}^{\ell} \phi(p_i^{\alpha_i}) = \prod_{i=1}^{\ell} p_i^{\alpha_i - 1}(p_i - 1)$ and is isomorphic to $\mathbb{Z}_{\phi(p_1^{\alpha_1})} \times \mathbb{Z}_{\phi(p_2^{\alpha_2})} \times \cdots \times \mathbb{Z}_{\phi(p_\ell^{\alpha_\ell})}$.*

Lemma 2. *Let $N > 1$ be any odd integer. If an integer e is relatively prime to $\phi(N)$, then the distribution $D_0(N, e) = \{r \leftarrow \mathbb{Z}_N^* : r\}$ is identical to the distribution $D_1(N, e) = \{r \leftarrow \mathbb{Z}_N^* : r^e\}$.*

Definition 9 (Roots and residues)**.** *Let N and e be any positive integers, and let $m \in \mathbb{Z}_N^*$. Let the set $Roots_{N,e}(m) = \{s \in \mathbb{Z}_N^* \mid s^e = m\}$. Let the set $Residues_{N,e} = \{m \in \mathbb{Z}_N^* \mid Roots_{N,e}(m) \neq \emptyset\}$.*

Lemma 3. *Let $p > 2$ be a prime number, and let $e > 2$ and $\alpha \geq 1$ be integers. Let $g = \gcd(e, p^{\alpha-1}(p-1))$. Then for any $m \in Residues_{p^\alpha,e}$, $|Roots_{p^\alpha,e}| = g$.*

Lemma 4. *Let $N > 1$ be an odd integer, and let $\prod_{i=1}^{\ell} p_i^{\alpha_i}$ be its prime factorization. Let $e > 1$ be an integer. Let $g_i = \gcd(e, p_i^{\alpha_i-1}(p_i - 1))$. Then for any $m \in Residues_{N,e}$, $|Roots_{N,e}(m)| = \prod_{i=1}^{\ell} g_i$.*

Lemma 5. *Let $N > 1$ be an odd integer, and $e > 1$ be an integer. Then r selected as follows is a uniformly random element of $\mathbb{Z}_N^*$: first, select y uniformly at random from $Residues_{N,e}$. Then, select r uniformly at random from $Roots_{N,e}(y)$.*

Lemma 6. *Let $N > 1$ be an odd integer, and $e > 1$ be an integer. Let $m \in Residues_{N,e}$. Let z be selected uniformly at random from $Residues_{N,e}$ and then picking one of its roots at random is equivalent to picking a random element of $\mathbb{Z}_N^*$.*

Lemma 7. *Let $N > 1$ be an odd integer, and $e > 1$ be an integer. Then for all $m \in Residues_{N,e}$, $z \in Residues_{N,e}$, $u \in Roots_{N,e}(z)$, the following outputs a uniformly random element of $Roots_{N,e}(m)$: pick $r \leftarrow Roots_{N,e}(z/m)$, output u/r.*

6.2.2 Proof of blindness

Let us consider $\mathcal{A}$'s interaction with the blindness challenger in the blindness game $\mathsf{Game}^{\text{blind}}_{E,\mathcal{A}}(\lambda)$, and then analyze what information $\mathcal{A}$ learns as a result of this interaction. When clear from context that integers in question are elements of $\mathbb{Z}_N^*$, we omit "mod N".

Since the Setup is done by the environment, we will assume that $\gcd(e, \phi(N)) = 1$. But we will keep in mind that if e is not relatively prime to $\phi(N)$ then perfect blindness could be compromised. This is discussed later on in section 6.2.3.

Claim 1. *If e is relatively prime to $\phi(N)$, then (m_{0_*}, y_0) and (m_{1_*}, y_1) (sent to $\mathcal{A}$ while it is its signing round in $\mathsf{Game}^{blind}_{E,\mathcal{A}}(\lambda)$) are both uniformly random elements of $\mathbb{Z}_N^* \times \mathbb{Z}_{n_1}$, and are distributed independently of b, m_0, m_1.*

Proof. The value R is sampled uniformly from the set $\mathbb{Z}_N^*$ then the uniformity of the distribution of $m_{0_*} = m_0 \cdot R_0^e \pmod N$ and $m_{1_*} = m_1 \cdot R_1^e \pmod N$ follows immediately from Lemma 2 in the set $\mathbb{Z}_N^*$. Then the uniformity of the distribution of $y_0 = m_{0_*}^{d_1'} \pmod{n_1}$ and $y_1 = m_{1_*}^{d_1'} \pmod{n_1}$ follows the same distribution in the set $\mathbb{Z}_{n_1}^*$ from Lemma 2 where $e = d_1'$ and $r = m_{i_*}$. □

Claim 2. *If e is relatively prime to $\phi(N)$, then $\mathcal{A}$'s view in the blindness game $\mathsf{Game}^{blind}_{E,\mathcal{A}}(\lambda)$ after receiving the signatures is independent of the bit b.*

Proof. $\mathcal{A}$ already knows whether it will receive the signatures or $\perp$, based on the values s_{*_b} and $s_{*_{\{1-b\}}}$ it sent to the challenger. If it receives the signatures, then there are unique values $R_{0,b}, R_{1,b}$ consistent with either $b \in \{0,1\}$, and they were equally likely to have been chosen (Lemma 2). If $\mathcal{A}$ does not receive the signatures, then $\mathcal{A}$ learns nothing. □

Claim 3. *If $\mathcal{A}$ chooses to drop the request or deactivate the client, then $\mathcal{A}$'s view in the blindness game $\mathsf{Game}^{blind}_{E,\mathcal{A}}(\lambda)$ is independent of b.*

Proof. $\mathcal{A}$ has the capability to drop the request by valuating the signature verification check as false. Alternatively, $\mathcal{A}$ can deactivate the client by either giving wrong clone-detection string, causing verification to fails in the next signing round, or just valuating the verification of the string as false. If any of the signing rounds don't conclude with a correct signature, then $\mathcal{A}$ does not receive the signatures. Then, if the adversary $\mathcal{A}$ chooses to drop the request or deactivate the client, $\mathcal{A}$ learns nothing. □

Theorem 1. *If both signatures verify, $\mathsf{VerifyPSS}(\sigma_0, \mathsf{msg}_0) = 1$ and $\mathsf{VerifyPSS}(\sigma_1, \mathsf{msg}_1) = 1$, then $\mathcal{A}$'s view in the blindness experiment $\mathsf{Game}^{blind}_{E,\mathcal{A}}(\lambda)$ is independent of b.*

Proof. Let us condition on the event that the signatures, on the two messages $\mathsf{msg}_0, \mathsf{msg}_1$ the adversary chooses, pass verification. In this case, the values $m_0 = H(\mathsf{msg}_0)$, $m_1 = H(\mathsf{msg}_1)$ computed by the challenger, as well as the values $m_{*_b} = m_b \cdot R_0^e \pmod N$, $m_{*_{1-b}} = m_{1-b} \cdot R_1^e \pmod N$ the challenger sent to the signer must all be in the set $Residues_{N,e}$. Let us consider a series of experiments.

Our first experiment is the case of running the blindness challenger with $b = 0$: The challenger begins by sampling m_0 and m_1 as PSS encodings of msg_0 and msg_1. Then, it samples $R_0 \leftarrow \mathbb{Z}_N^*, R_1 \leftarrow \mathbb{Z}_N^*$, computes $m_{*_0} = m_0 \cdot R_0$ and $m_{*_1} = m_1 \cdot R_1$, and sends $(m_*, m'_*) = (m_{*_0}, m_{*_1})$ to the adversary. The adversary responds with (s_{*_0}, s_{*_1}), and the challenger computes the signatures $\sigma_0 = s_{*_0}/R_0$ and $\sigma_1 = s_{*_1} \cdot R_1$.

By Lemma 5, instead of choosing R_0 and R_1 uniformly at random from $\mathbb{Z}_N^*$ and then setting $m_{*_0} = R_0^e \cdot m_0 \pmod N$ and $m_{*_1} = R_1^e \cdot m_1 \pmod N$, one could equivalently choose x_0, x_1 uniformly at random from $Residues_{N,e}$, and then let $m_{*_0} = x_0 \cdot m_0 \pmod N$, $m_{*_1} = x_1 \cdot m_1 \pmod N$, $R_0 \leftarrow Roots_{N,e}(x_0)$, $R_1 \leftarrow Roots_{N,e}(x_1)$; let us call the resulting experiment A_0.

By Lemma 6, this is equivalent to choosing m_{*_0} and m_{*_1} uniformly at random from $Residues_{N,e}$, and letting $R_0 \leftarrow Roots_{N,e}(m_{*_0}/m_0)$, $R_1 \leftarrow Roots_{N,e}(m_{*_1}/m_1)$; let us call the resulting experiment B_0.

By Lemma 7, this is equivalent to picking m_{*_0} and m_{*_1} uniformly at random from $Residues_{N,e}$ and sending the adversary the pair $(m_*, m'_*) = (m_{*_0}, m_{*_1})$, and upon receipt of s_{*_0} and s_{*_1} such that $s_{*_0}^e = m_{*_0}$ and $s_{*_1}^e = m_{*_1}$, outputting $\sigma_0 \leftarrow Roots_{N,e}(m_0)$, $\sigma_1 \leftarrow Roots_{N,e}(m_1)$; let us call the resulting experiment C_0.

Let us obtain a new experiment, C_1, by modifying C_0: let $(m_*, m'_*) = (m_{*_1}, m_{*_0})$, while everything else stays the same. C_1 gives the adversary identical view to C_0. Let

B_1 be the same as B_0 except for $(m_*, m'_*) = (m_{*_1}, m_{*_0})$; by Lemma 7, the adversary's view here is identical to C_1. Let A_1 be identical to A_0 except $(m_*, m'_*) = (m_{*_1}, m_{*_0})$; by Lemma 6, it is identical to B_1. Finally, by Lemma 5, A_1 gives the adversary the same view as the challenger when $b = 1$. □

From the Claims 1,2,3 and Theorem 1 we can see that when e is relatively prime to $\phi(N)$ the adversary $\mathcal{A}$ receives the same view in the blindness game $\mathsf{Game}^{\text{blind}}_{E,\mathcal{A}}(\lambda)$ for $b = 0$ as for $b = 1$. Therefore the advantage of the adversary $\mathsf{Adv}^{\text{blind}}_{E,\mathcal{A}}(\lambda) = 0$ and we can say that RSA-BSK is perfectly blind by Definition 5.

6.2.3 Discussion on malicious modulus generation

In the paper "Security of RSA-BSSA"[Lys22] it is discussed what would happen to the blindness of an RSA-PSS based blind signature scheme if the public exponent e is not relatively prime to $\phi(N)$ and a formula is given. Let us discuss this further with some numbers from the RSA-BSK protocol. Let us assume that the adversary will generate their public modulo maliciously. The formula that was given is: $N = \prod_{i=1}^{\ell} p_i$ such that $e \mid p_i - 1$ for $1 \leq i \leq \ell$, where each p_i is a distinct prime number, m_{*_b} reveals $\ell \cdot \log e$ bits of information. Then the public modulo for the adversary could be described with $n_2 = \prod_{i=1}^{\ell} p_i$ with primes such that $e \mid p_i - 1$ for $1 \leq i \leq \ell$, where each p_i is a distinct prime number. We will use $e = 65537 = 2^{16} + 1$, since that is the standard value for e. Let us assume $N \approx 6144$ bits $(3072 + 3072)$, where $n_1, n_2 \approx 3072$ bits. From that we can assume that only n_2 is generated maliciously, since the server acts as an adversary against blindness. To leak the most amount of information that a maliciously generated moduli could, it needs to be composed of the maximum amount of primes that meet the aforementioned requirements. So the primes that could be picked are in the set $P(e) = \{a \in \mathbb{Z}^+, p \in \mathbb{P} \mid p = 2 \cdot e \cdot a + 1\}$. We calculated the biggest set that had a product of less than 3072 bits, with the smallest primes in $P(e)$, we present the set with values a of the values that fit that product:

{7, 10, 13, 18, 27, 33, 40, 42, 45, 52, 69, 84, 102, 103, 109, 114, 117, 145, 147, 189, 207, 210, 213, 217, 223, 228, 244, 249, 250, 262, 264, 265, 279, 280, 289, 304, 312, 313, 319, 340, 343, 345, 352, 354, 355, 360, 364, 367, 373, 384, 390, 415, 420, 433, 438, 447, 459, 475, 492, 495, 507, 517, 520, 522, 529, 544, 553, 555, 570, 592, 607, 615, 634, 637, 685, 700, 702, 712, 720, 742, 747, 748, 753, 774, 783, 795, 805, 823, 829, 844, 849, 868, 870, 877, 889, 894, 895, 903, 907, 913, 915, 924, 937, 943, 945, 955, 958, 978, 979, 982, 984, 994, 1000, 1003, 1005, 1012, 1027, 1032, 1042, 1047}

From this we can conclude that there are at maximum 120 prime numbers that could suit our purposes. According to the previously introduced formula, we get for $\ell = 120,\ e = 2^{16} + 1$ the answer $1920 \approx 120 \cdot \log(2^{16} + 1)$ bits of information leaked. This

is about $31.25\% = 1920/6144$ of the entire message m_*. This leaked information is in the information-theoretic sense, thus we can not conclude if this is enough information to beat a singular instance of the blindness game $\mathsf{Game}^{\text{blind}}_{E,\mathcal{A}}(\lambda)$. But is enough to beat the game more times than not, breaking perfect blindness.

One way to mitigate this kind of attack would be to test signing on a random message after the moduli is generated. If the signature verifies with the message then the public moduli was not picked maliciously. Since if $\gcd(\phi(N), e) \neq 1$, then there does not exist a value d, such that $d \equiv e^{-1} \pmod{\phi(N)}$ and therefore there is no value such that $m \equiv m^{d \cdot e} \pmod{N}$.

RSA-BSSA [Lys22] has two versions of the protocol to prevent this sort of attack. Version A includes generating a proof in the key generation algorithm and stores it in the public key of the signer that proves that e is relatively prime to $\phi(N)$. This is necessary in RSA-BSSA since there is only one signing party and they have the whole control over the public exponent. This is not a problem in RSA-BSK, because the public exponent is computed by two parties and fixed at the end of the key generation process. Therefore there is no need to prove at the beginning of every signing process that e is relatively prime to $\phi(N)$. It is only necessary to check once if e is relatively prime to $\phi(N)$ in the form of a signing test. Version B ensures perfect blindness by not signing a message directly, instead invoking signing on a message that is already obscured. By doing so a malicious party can not reveal the original message even if they manage to completely remove the blindness from the message. Since the original message is obscured in a way that is not reversible without knowing how it was obscured. Although this also creates unwanted clutter. Since the signature can not be verified on the original message, but rather on the obscured version of it. This is a unwanted drawback of Version B that is not wanted in the protocol of RSA-BSK.

6.3 One-more unforgeability of RSA-BSSA

We will be using the definition of RSA-ACTI as written in "Security Analysis of RSA-BSSA" [Lys22].

Definition 10 (RSA-ACTI [BNPS03, Lys22])**.** *Let $\mathcal{A}$ be an oracle Turing machine. For the security parameter k, let the experiment $\mathsf{Exp}^{\mathsf{rsa-acti}}_{\mathcal{A}}(k)$ be defined as follows:*

RSA key pair generation *The challenger generates an RSA public key $(N.e)$ and secret key d corresponding to the security parameter k. Let us define the following oracles:*

1. *The RSA inversion oracle $\mathcal{O}_I(\cdot, N, d)$ that, on input $y \in \mathbb{Z}^*_N$, returns x such that $x = y^d \pmod{N}$ where $e \cdot d \equiv 1 \pmod{\phi(N)}$.*

2. *An oracle $\mathcal{O}_R(\cdot, N)$ that, when queried, issues a random RSA inversion challenge point, i.e. a random element of $\mathbb{Z}_N^*$. By y_i, let us denote the outcome of the i^{th} such query.*

$\mathcal{A}$ is invoked *The challenger invokes $\mathcal{A}^{\mathcal{O}_I(\cdot,N,d),\mathcal{O}_R(\cdot,N)}(N, e)$ and responds to its oracle queries. Eventually, $\mathcal{A}$ terminates.*
$\mathcal{A}$'s success criterion *Let ℓ be the number of queries $\mathcal{A}$ issued to $\mathcal{O}_I(\cdot, N, d)$. Let $(y_1, \ldots, y_n)$ be the values $\mathcal{A}$ received from $\mathcal{O}_R(\cdot, N)$. Let $(z_1, \ldots, z_n)$ be $\mathcal{A}$'s output. For $1 \leq i \leq n$, z_i is correct if $z_i^e = y_i \pmod N$. $\mathcal{A}$ is successful if $|\{i : z_i \text{ is correct}\}| \geq \ell + 1$.*

By $\mathsf{Adv}_{\mathcal{A}}^{rsa-acti}(k)$ we denote the probability that $\mathcal{A}$ is successful in $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{rsa-acti}}(k)$. The RSA-ACTI problem is hard if for any PPT $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{rsa-acti}(k)$ is negligible.

$\mathsf{Game}_{E,\mathcal{A}}^{\text{somuf}}(\lambda)$

$\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \{\mathsf{KeyGen}_j(\mathsf{par})\}_{j=1}^n$
$\ell \leftarrow 0$
$S_1, \ldots, S_r \leftarrow \emptyset$
$Q \leftarrow \mathcal{A}^{\mathsf{SignO}}(\mathsf{pk})$
return $\perp$ **if** $Q = \emptyset$
for $i \in |Q|$:
  if $\mathsf{VerifyPSS}(\mathsf{pk}, m_i, \sigma_i) = 0$
    return $\perp$
for all $k \in |Q|, i \in |Q|, k \neq i$:
  return $\perp$ **if** $(m_k, \sigma_k) = (m_i, \sigma_i)$
if $|Q| \geq \ell + 1$:
  return 1
return $\perp$

$\mathsf{SignO}(m)$

return $\perp$ **if** $m \in S_1$
$S_1 \leftarrow S_1 \cup m$
$(m, \sigma) \leftarrow \mathsf{Sign}_1(m, \mathsf{sk})$
$\ell \leftarrow \ell + 1$
return (m, σ)

Figure 5. Game for strong one-more unforgeability of RSA-BSSA

RSA-BSSA is strong one-more unforgeable in the random-oracle model under the assumption that RSA-ACTI problem is hard [Lys22]. We will be under the same assumption in our strong one-more unforgeability proof of RSA-BSK.

6.4 Security against malicious servers

As in [BKLO17], we consider a malicious server as an adversary $\mathcal{A}$ that has a share d_1'' of the client's private modulus d_1, the server's private modulus d_2, and also has a connection to client's signature device that sends signing requests to the server. We can represent these values as such $\mathsf{sk}_C = d_1', \mathsf{sk}_S = (d_1'', d_2)$ in the game $\mathsf{Game}^{\text{somuf-S}}_{E,\mathcal{A}}(\lambda)$. We assume that $\mathcal{A}$ is able to use such a connection as an oracle SignO i.e. to choose messages m, send m to the oracle and obtain $\mathsf{SignO} = P(m)^d \pmod n$. The adversary wants to generate (σ, m) pair that was not queried through the oracle. If it manages to create a signature on more messages than it has queried, then the adversary has broken strong one-more unforgeability of the server side.

$\underline{\mathsf{Game}^{\text{somuf-S}}_{E,\mathcal{A}}(\lambda)}$

$\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{pk}, \mathsf{sk}_C, \mathsf{sk}_S) \leftarrow \{\mathsf{KeyGen}_j(\mathsf{par})\}_{j=1}^n$
$\ell \leftarrow 0$
$S_1, \ldots, S_r \leftarrow \emptyset$
$Q \leftarrow \mathcal{A}^{\mathsf{SignO}_1, \{\mathsf{SignO_j}\}_{j=2}^{r-1}, \mathsf{SignO_r}}(\mathsf{sk}_S, \mathsf{pk})$
return $\perp$ **if** $Q = \emptyset$
for $i \in |Q|$:
  if $\mathsf{VerifyPSS}(\mathsf{pk}, m_i, \sigma_i) = 0$
    return $\perp$
for all $k \in |Q|, i \in |Q|, k \neq i$:
  return $\perp$ **if** $(m_k, \sigma_k) = (m_i, \sigma_i)$
if $|Q| \geq \ell + 1$:
  return 1
return $\perp$

$\underline{\mathsf{SignO}_1(m)}$

return $\perp$ **if** $m \in S_1$
$S_1 \leftarrow S_1 \cup m$
$(\mathsf{st}_{1,m}, m) \leftarrow \mathsf{Sign}_1(m, \mathsf{sk}_j)$
return $(\mathsf{st}_{i,m}, m)$

$\underline{\mathsf{SignO}_j(m, \mathsf{st}_{j-1,m})}$

return $\perp$ **if** $m \notin S_1, \ldots, S_{j-1}$
return $\perp$ **if** $m \in S_j$
$S_j \leftarrow S_j \cup \{m\}$
$(\mathsf{st}_{j,m}, m) \leftarrow \mathsf{Sign}_r(\mathsf{st}_{j-1,m}, \mathsf{sk}_j)$
return $(\mathsf{st}_{j,m}, m)$

$\underline{\mathsf{SignO}_r(m, \mathsf{st}_{r-1,m})}$

return $\perp$ **if** $m \notin S_1, \ldots, S_{r-1}$
return $\perp$ **if** $m \in S_r$
$S_r \leftarrow S_r \cup \{m\}$
$(m, \sigma) \leftarrow \mathsf{Sign}_r(\mathsf{st}, \mathsf{sk}_r)$
$\ell \leftarrow \ell + 1$
return (m, σ)

Figure 6. Game for strong one-more unforgeability against a malicious server

Lemma 8. *The distribution of value d'_1 that is obtained through the genShare function is indistinguishable from being uniformly distributed in the set $\mathbb{Z}^*_{\phi(n_1)}$.*

Proof. We can sample d'_1 randomly from the set $\mathbb{Z}^*_{\phi(n_1)}$ or generate it through genShare. Since genShare algorithm relies on a pseudo-random function (PRF) which is a bijection, the distribution of the value d'_1 in both cases is identical. [SVL24b] □

Theorem 2. *If there exists an adversary $\mathcal{B}$ that has an advantage $p_{\mathcal{B}}(k)$ against the malicious server strong one-more unforgeability game $\mathsf{Game}^{\textit{somuf-S}}_{E,\mathcal{A}}(\lambda)$ of RSA-BSK then there also exists an adversary $\mathcal{A}$ that has an advantage $p_{\mathcal{A}}(k)$ against the strong one-more unforgeability game of RSA-BSSA, where $p_{\mathcal{A}}(k) \approx p_{\mathcal{B}}(k)$.*

Proof. From Lemma 8 we know that genShare$^{\Phi}$ returns a uniformly distributed number d'_1 in the context of the server, then it doesn't give any advantage to the adversary in regards to it. If d'_1 is uniformly distributed then $d''_1 = d_1 - d'_1 \pmod{\phi(n_1)}$ is also uniformly distributed and gives no information about d_1, without having access to the value d'_1. From this we can see that d''_1 holds no value to an adversary without access to d'_1. Then if the operation of computing m^{d_1} is done in the signing oracle, and not by the server, changes only the advantage of the adversary by computational time of the operation $s_1 = y \cdot m^{d''_1} \pmod{n_1}$ which is negligibly small.
With these changes we can see that the adversary needs to either compute d_1 to get (m^{d_1}, s_1) or compute (m^{d_1}, s_1) that wasn't queried under the oracle. This is negligibly different from the strong one-more unforgeability game in RSA-BSSA, since in the game operations are computationally similar, and inputs and outputs are the same. For completeness we can construct adversary $\mathcal{A}$ that breaks strong one-more unforgeability of RSA-BSSA with an upper bound on its running time represented as $t^{\mathcal{A}}(k)$, and $p_{\mathcal{A}}(k)$ as its success probability. Then for the adversary $\mathcal{B}$, that breaks strong one-more unforgeability of RSA-BSK on server side, we can compute the upper bound on its running time $t^{\mathcal{B}}(k)$ as such:
$t^{\mathcal{B}}(k) + t(t_r + t_{composite} + t_{verify} + t_{lookup} + t_{ex}) \geq t^{\mathcal{A}}(k)$, where t_r is time it takes to verify r and generate a new one, $t_{composite}$ is the time to generate the composite signature, t_{verify} is the time to verify the client signature, t_{lookup} is the time to lookup the values of the particular client, t_{ex} is the time of one exponentiation. Since $t^{\mathcal{A}}(k) \gg t_r + t_{composite} + t_{verify} + t_{lookup} + t_{ex}$ then $t^{\mathcal{B}}(k) \approx t^{\mathcal{A}}(k)$. Then the probability of success for adversary $\mathcal{B}$ that wins in the strong one-more unforgeability against malicious server game $\mathsf{Game}^{\text{somuf-S}}_{E,\mathcal{A}}(\lambda)$ is $p_{\mathcal{B}}(k) \approx p_{\mathcal{A}}(k)$. From this we can derive that if RSA-BSK is strong one-more unforgeable against malicious server then RSA-BSSA is also strong one-more unforgeable. □

6.5 Security against malicious client

The strong one-more unforgeability game against a malicious client $\mathsf{Game}^{\text{somuf-C}}_{E,\mathcal{A}}(\lambda)$ is almost the same as the malicious server game $\mathsf{Game}^{\text{somuf-S}}_{E,\mathcal{A}}(\lambda)$, but instead of the adversary

$\mathcal{A}$ having access to the server's secret key $\mathsf{sk}_S = (d_1'', d_2)$, it only has access to the client's secret key $\mathsf{sk}_C = d_1'$.

$\mathsf{Game}^{\text{somuf-C}}_{E,\mathcal{A}}(\lambda)$

$\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{pk}, \mathsf{sk}_C, \mathsf{sk}_S) \leftarrow \{\mathsf{KeyGen}_j(\mathsf{par})\}_{j=1}^{n}$
$\ell \leftarrow 0$
$S_1, \ldots, S_r \leftarrow \emptyset$
$Q \leftarrow \mathcal{A}^{\mathsf{SignO}_1, \{\mathsf{SignO}_j\}_{j=2}^{r-1}, \mathsf{SignO}_r}(\mathsf{sk}_C, \mathsf{pk})$
return $\perp$ **if** $Q = \emptyset$
for $i \in |Q|$:
  if $\mathsf{VerifyPSS}(\mathsf{pk}, m_i, \sigma_i) = 0$
    return $\perp$
for all $k \in |Q|, i \in |Q|, k \neq i$:
  return $\perp$ **if** $(m_k, \sigma_k) = (m_i, \sigma_i)$
if $|Q| \geq \ell + 1$:
  return 1
return $\perp$

$\mathsf{SignO}_1(m)$

return $\perp$ **if** $m \in S_1$
$S_1 \leftarrow S_1 \cup m$
$(\mathsf{st}_{1,m}, m) \leftarrow \mathsf{Sign}_1(m, \mathsf{sk}_j)$
return $(\mathsf{st}_{i,m}, m)$

$\mathsf{SignO}_j(m, \mathsf{st}_{j-1,m})$

return $\perp$ **if** $m \notin S_1, \ldots, S_{j-1}$
return $\perp$ **if** $m \in S_j$
$S_j \leftarrow S_j \cup \{m\}$
$(\mathsf{st}_{j,m}, m) \leftarrow \mathsf{Sign}_r(\mathsf{st}_{j-1,m}, \mathsf{sk}_j)$
return $(\mathsf{st}_{j,m}, m)$

$\mathsf{SignO}_r(m, \mathsf{st}_{r-1,m})$

return $\perp$ **if** $m \notin S_1, \ldots, S_{r-1}$
return $\perp$ **if** $m \in S_r$
$S_r \leftarrow S_r \cup \{m\}$
$(m, \sigma) \leftarrow \mathsf{Sign}_r(\mathsf{st}, \mathsf{sk}_r)$
$\ell \leftarrow \ell + 1$
return (m, σ)

Figure 7. Game for strong one-more unforgeability against a malicious client

Claim 4. *The usage of a different verification function than the verification of the RSA-BSK signature on the server side does not affect the outcome of the strong one-more unforgeability game* $\mathsf{Game}^{\textit{somuf-C}}_{E,\mathcal{A}}(\lambda)$.

Proof. The verification function on the server side is used to verify the authenticity of the value of d_1'. Therefore it does not need to check the correctness of the message encoding. we can see that this does not change the outcome of the game since RSA-BSSA is strong one-more unforgeable and there is no check for correct signature on the server side. □

Still in the end the signature needs to be PSS-encoded to break strong one-more unforgeability, same as in RSA-BSK, since RSA-BSK messages also need to be PSS-encoded.

Lemma 9. *An adversary in the malicious client strong one-more unforgeability game* $\mathsf{Game}^{\textit{somuf-C}}_{E,\mathcal{A}}(\lambda)$ *can not obtain non-negligible advantage through the Signing oracle with dropping requests or deactivating clients.*

Proof. If the adversary doesn't use the correct r then the client is deactivated and $\mathcal{A}$ therefore loses advantage, since an deactivated client can not request for signatures. Same goes if the adversary gives wrong d_1' to the server T_0 in a row. Also giving the incorrect d_1' to the server doesn't give information to the adversary, since the server drops the request. Only if the adversary gives an d_1' value that verifies does the server give an signature back, but d_1' then needs to be the correct one or $d_1' + \phi(n_1) \cdot a \equiv d_1' \pmod{\phi(n_1)}$, where $a \in \mathbb{Z}$, from which one could calculate d_1, but this is more negligible than finding d_1 separately, since operations are more costly. From this we can conclude that the adversary has the best advantage if they follow the protocol requirements. □

Theorem 3. *If RSA-BSK is strong one-more unforgeable against malicious client, then RSA-BSSA is also strong one-more unforgeable.*

Proof. Adversary $\mathcal{A}$ is playing the strong one-more unforgeable against malicious client game $\mathsf{Game}^{\text{somuf-C}}_{E,\mathcal{A}}(\lambda)$ of RSA-BSK. Let us construct an adversary $\mathcal{B}$ that plays the strong one-more unforgeability game of RSA-BSSA and has access to the public key (n_2, e). $\mathcal{B}$ samples two primes $p_1, q_1 \xleftarrow{\$} S_{(\ell,s)}$ and computes $n_1 = p_1, q_1$ and $d_1 = e^{-1} \pmod{\phi(n_1)}$, then sends $\mathcal{A}$ the value (N, n_1, e), where $N = n_1 \cdot n_2$. $\mathcal{A}$ declares (N, n_1, e) as its public key.
When $\mathcal{A}$ wants to get a signature on a message m the process follows as such:

- $\mathcal{A}$ sends $\mathcal{B}$ the message m.
- $\mathcal{B}$ invokes its signing oracle with the message m and gets returned a signature s_2.
- $\mathcal{B}$ computes $s_1 = m^{d_1} \pmod{n_1}$, after which $\mathcal{B}$ computes $\sigma = C_{n_1,n_2}(s_1, s_2)$.
- $\mathcal{B}$ then sends (m, σ) to $\mathcal{A}$.

When adversary $\mathcal{A}$ manages to create a message-signature pair (m, σ) that verifies and was not queried through the signing oracle, they send (m, s_1) to $\mathcal{B}$, where $s_1 = \sigma \pmod{n_1}$. $\mathcal{B}$ then proceeds to verify every message-signature pair that went through $\mathcal{B}$. If the count of message-signature pairs that pass verification is larger than the count of times $\mathcal{B}$ used their signing oracle, they send 1 to $\mathcal{A}$, otherwise they send $\perp$. If $\mathcal{A}$ receives 1 they win, otherwise lose. If $\mathcal{A}$ wins then $\mathcal{B}$ also wins.

From this we can derive that the upper bound running time for beating strong one-more unforgeability against a malicious client game $\mathsf{Game}^{\text{somuf-C}}_{E,\mathcal{A}}(\lambda)$ by taking the upper bound on the running time of adversary $\mathcal{A}$ as $t^{\mathcal{A}}(k) \geq t^{\mathcal{B}}(k)$, where $t^{\mathcal{B}}(k)$ is the upper bounds on the running time for $\mathcal{B}$. From this we derive that $p_{\mathcal{A}}(k) \approx p_{\mathcal{B}}(k)$. Therefore if

there exist an adversary that breaks strong one-more unforgeability against a malicious client in the RSA-BSK protocol, then there exists an adversary that can break one-more unforgeability of RSA-BSSA. Then if RSA-BSK is strong one-more unforgeable against a malicious client then RSA-BSSA is also strong one-more unforgeable. □

7 Future Work

7.1 Weaknesses of blind signatures

Since the server does not know the message that they are signing (proved in Section 6.2) then it is not possible to verify if the message was queried through the server or not. That means if an adversary manages to create a valid signature without querying the server, then there is no way of disproving the signature. A way of detecting such forged signatures is to reveal some part of the message during the signing rounds. For instance if the message has an timestamp then you can cross-reference with the server if an signing round did occur at such time. This example can also not be entirely correct since if an adversary knows of an message that was singed at that timestamp then they can cover their tracks by taking that same timestamp. Then the only way of showing that it might be invalid is to show the other signature with the same timestamp, in doing so both signatures may be assumed to be invalid. All of this is assuming that the server keeps records of signing rounds and its outcomes. This way some or all of the blindness attribute can be lost, so it is not recommended if an blind signature scheme is to be implemented.

A way to fix this weakness would be to add fail-stop signatures [Pfi91, Ble11] to the blind signature scheme. Fail-stop signatures provide unconditional unforgeability by allowing verification of whether a signature was generated by a trusted signer or not. Although this could provide a security guarantee that a signature was not forged, little research [CLS05, CCYC14] has been conducted on fail-stop signatures in the context of blind signature schemes.

7.2 Implementation

If RSA-BSK were to be implemented on a system these are some things to consider:

- The verification algorithm of a signature in the signing process is different from the verification algorithm of a signature that has completed the signing process (Verify and VerifyPSS), discussed in Section 5.
- It is required to control if the public exponent N has been computed maliciously e.g. if it is relatively prime to the public exponent e. We discussed a way of doing this with a signing test in Section 6.2.3.

8 Conclusion

In this work, a new blind signature scheme was designed, named RSA-BSK. In the security analysis we proved that it is perfectly blind if the Setup and KeyGen is done non-maliciously, strong one-more unforgeable against malicious client and server, and the composition procedure is correct. The scheme was designed to induce minimal necessary changes for implementation on systems that already use SplitKey. Also it was discussed how much of a blind message could be leaked if the public modulus would be picked maliciously and how to mitigate this attack.

In the future it would be necessary to introduce security elements that detect if public moduli have been computed maliciously. Right now it is recommended to do a test signing before sending anything private, since if the public moduli has been computed maliciously then the signature won't verify. Also a verification function that would be the same in signing process as it is in the verification process. Lastly it could be researched if there is a way to add security elements to the signature that detect if it was computed by a trusted party or an malicious party. One such solution is discussed with the addition of fail-stop signatures.

A way to add detection of maliciously generated public moduli in the protocol is already being researched.

In conclusion, RSA-BSK is the first secure blind signature scheme using the functionalities of SplitKey. Though it needs further research and guide how to implement it.

Acknowledgments. I would like to thank Nikita for giving me guidance, study material, and support through the entire duration of writing this Bachelor's thesis. I would also like to thank Peeter for his discussions on the topic and insight into security proofs. Lastly I give a thanks to Toomas and Semjon for helping me with the structure and wording of the thesis.

References

[BKLO17] Ahto Buldas, Aivo Kalu, Peeter Laud, and Mart Oruaas. Server-supported rsa signatures for mobile devices. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, Computer Security – ESORICS 2017, pages 315–333, Cham, 2017. Springer International Publishing.

[Ble11] Gerrit Bleumer. Fail-Stop Signature, pages 446–447. Springer US, Boston, MA, 2011.

[BNPS03] Bellare, Namprempre, Pointcheval, and Semanko. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. Journal of Cryptology, 16(3):185–215, Jun 2003.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures-how to sign with rsa and rabin. In Ueli Maurer, editor, Advances in Cryptology — EUROCRYPT '96, pages 399–416, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[CCYC14] Kai Chain, Jonathan Jen-Rong Chen, Jar-Ferr Yang, and Kuei Hu Chang. An improved fail-stop signature scheme based on dual complexities. International Journal of Innovative Computing, Information and Control, 10(2):535–544, 2014.

[Cha82] David Chaum. Blind signatures for untraceable payments. In Advances in Cryptology: Proceedings of CRYPTO '82, pages 199–203. Plenum, 1982.

[CKM+23] Elizabeth Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Snowblind: A threshold blind signature in pairing-free groups. Cryptology ePrint Archive, Paper 2023/1228, 2023.

[CLS05] Henry Ker-Chang Chang, Erl-Huei Lu, and Pin-Chang Su. Fail-stop blind signature scheme design based on pairings. Applied Mathematics and Computation, 169(2):1324–1331, 2005.

[Des88] Yvo Desmedt. Society and group oriented cryptography: a new concept. In Carl Pomerance, editor, Advances in Cryptology — CRYPTO '87, pages 120–127, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.

[DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, Advances in Cryptology — CRYPTO' 89 Proceedings, pages 307–315, New York, NY, 1990. Springer New York.

[DMS15] Ivan Damgård, Gert Læssøe Mikkelsen, and Tue Skeltved. On the security of distributed multiprime rsa. In Jooyoung Lee and Jongsung Kim, editors, Information Security and Cryptology - ICISC 2014, pages 18–33, Cham, 2015. Springer International Publishing.

[Hin08] M. Hinek. On the security of multi-prime rsa. Journal of Mathematical Cryptology, 2, 09 2008.

[KM14] Veronika Kuchta and Mark Manulis. Rerandomizable threshold blind signatures. In Revised Selected Papers of the 6th International Conference on Trusted Systems - Volume 9473, INTRUST 2014, page 70–89, Berlin, Heidelberg, 2014. Springer-Verlag.

[KRB+24] Ioanna Karantaidou, Omar Renawi, Foteini Baldimtsi, Nikolaos Kamarinakis, Jonathan Katz, and Julian Loss. Blind multisignatures for anonymous tokens with decentralized issuance. In Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS '24, page 1508–1522, New York, NY, USA, 2024. Association for Computing Machinery.

[LNÖ25] Anja Lehmann, Phillip Nazarian, and Cavit Özbay. Stronger security for threshold blind signatures. In Serge Fehr and Pierre-Alain Fouque, editors, Advances in Cryptology – EUROCRYPT 2025, pages 335–364, Cham, 2025. Springer Nature Switzerland.

[Lys22] Anna Lysyanskaya. Security analysis of RSA-BSSA. Cryptology ePrint Archive, Paper 2022/895, 2022.

[MOV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. Handbook of Applied Cryptography (1st ed.). Boca Raton, 1997.

[Pfi91] Birgit Pfitzmann. Fail-stop signatures: Principles and applications. In Proc. Compsec, volume 91, pages 125–134. Citeseer, 1991.

[SVL24a] Nikita Snetkov, Jelizaveta Vakarjuk, and Peeter Laud. TOPCOAT: towards practical two-party Crystals-Dilithium. Discover Computing, 27(1):18, Jul 2024.

[SVL24b] Nikita Snetkov, Jelizaveta Vakarjuk, and Peeter Laud. Universally Composable Server-Supported Signatures for Smartphones. Cryptology ePrint Archive, Paper 2024/1941, 2024.

[VZK03] Duc Liem Vo, Fangguo Zhang, and Kwangjo Kim. A new threshold blind signature scheme from pairings. In SCIS2003, pages 233–238. SCIS, 2003.

[Wil23] Jan Willemson. Analyzing and improving eligibility verifiability of the proposed belgian remote voting system. In Ruben Rios and Joachim Posegga, editors, Security and Trust Management, pages 126–135, Cham, 2023. Springer Nature Switzerland.

[WOW08] Zooko Wilcox-O'Hearn and Brian Warner. Tahoe: the least-authority filesystem. In Proceedings of the 4th ACM International Workshop on Storage Security and Survivability, StorageSS '08, page 21–26, New York, NY, USA, 2008. Association for Computing Machinery.

Appendices

A. Visual representation of processes

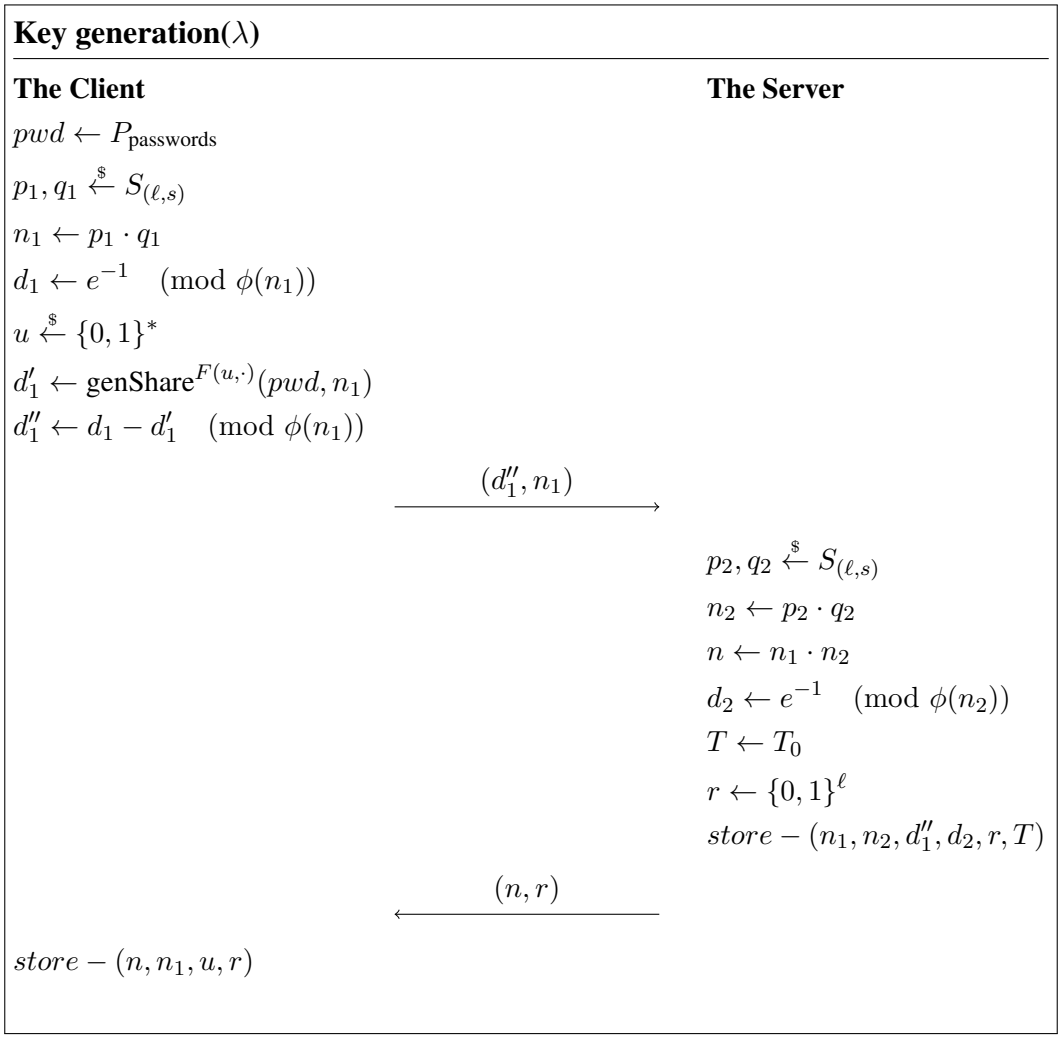

Figure 8. Visual representation of the key generation of SplitKey

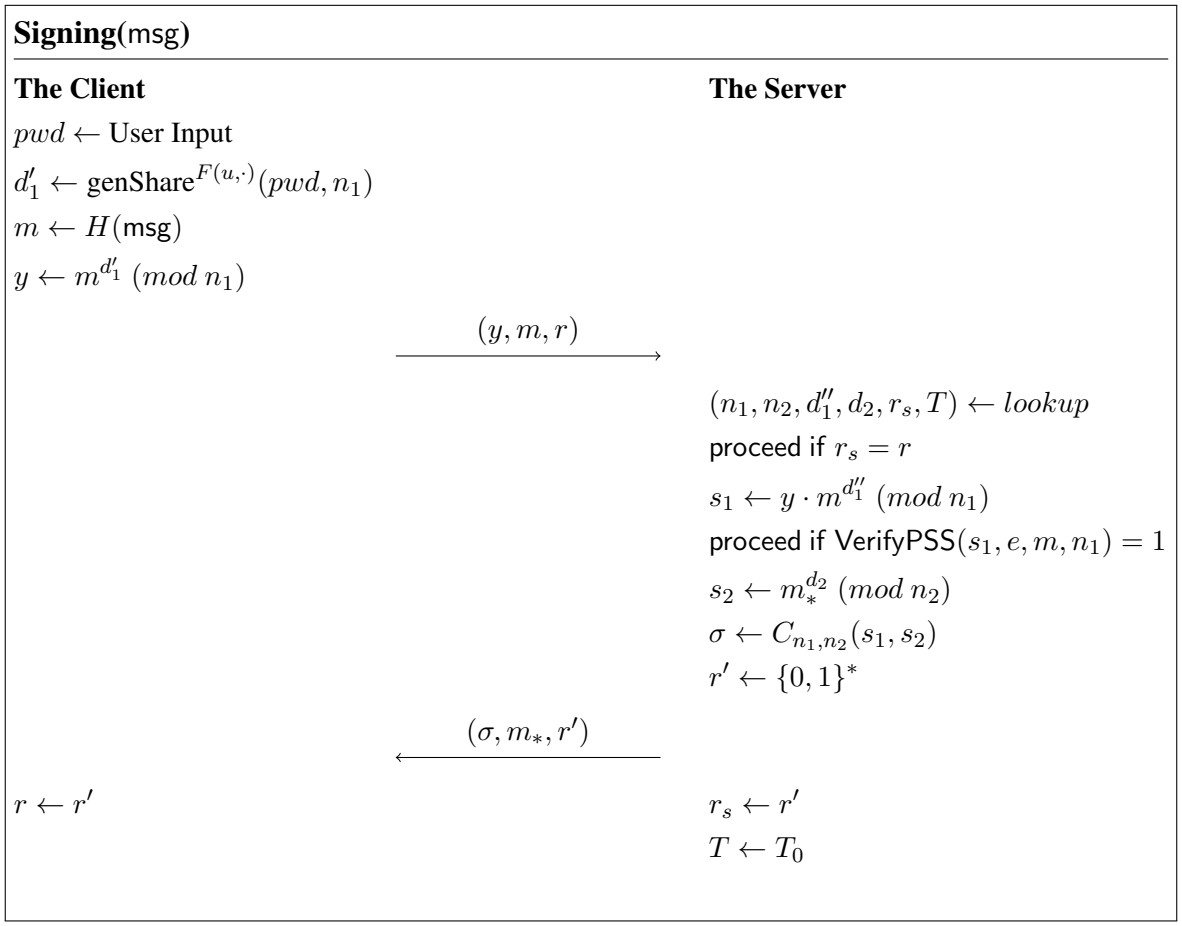

Figure 9. Visual representation of the signing process of SplitKey

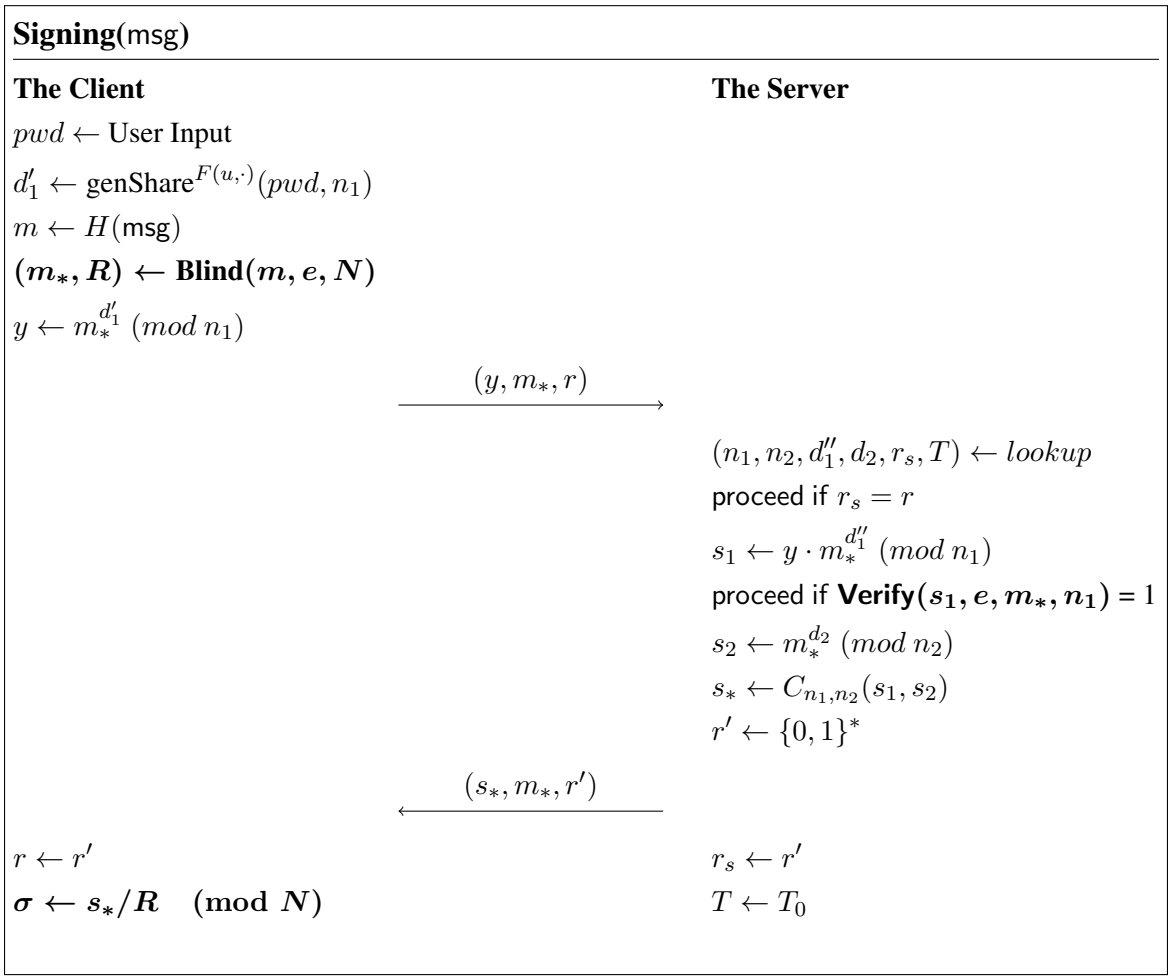

Figure 10. Visual representation of the signing process of RSA-BSK

B. License

Non-exclusive licence to reproduce the thesis and make the thesis public

I, **Mihkel Jaas Karu**,

1. grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright, my thesis **Design and Security Analysis of Blind Smart-ID/SplitKey Signature,** supervised by Nikita Snetkov, Peeter Laud and Toomas Krips;

2. grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright;

3. am aware of the fact that the author retains the rights specified in points 1 and 2;

4. confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Mihkel Jaas Karu
2025-05-15