Threshold PS Signature Scheme

Bingsen Wang, 201903975

Master's Thesis, Computer Science
June 2021
Advisor: Claudio Orlandi

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Abstract

Threshold signature scheme and blind signature plays a key role in cryptograph as well as in practical applications as e-Cash and e-Voting systems. Pointcheval and Sanders (PS) proposed a efficient randomizable blind signature scheme supporting multiple message based on pairing-based cryptography. In this work, we construct threshold blind signature scheme based on PS signature scheme, We use several different methods to do this, based on different secret sharing scheme and share conversion. As time efficiency is really important when it comes to pairing based cryptography, we also implement and test our proposed scheme based on BLS12-381 curve, which is known to be the most efficient and mature implementation of pairing friendly curve designed by Zcash. At the end of this work, we will also discuss several application scenarios that can exploit the good property of threshold blind signature scheme.

Keywords: Threshold signature, Blind signature, Pairing based cryptography, BLS curve, Pointcheval-Sanders signature scheme.

Acknowledgments

When the epidemic was raging, I left Denmark in early April and complete my master's thesis in my hometown. Here, I would first like to thank professor Claudio Orlandi, he advised my thesis through online methods, and carefully helped me find errors and problems in my thesis. Professor Diego F. Aranha, who is an expert in implementing cryptosystem, he helped me find the best implementation of pairing based cryptography which saved a lot of time for me.

During the two years of studying for a master's degree in computer science at Aarhus University, because I have always been interested in theoretical computer science, especially cryptography. I took three modules of theoretical computer science, including algorithms, cryptography and programming language. As more and more computer schools are putting more resources in the field of artificial intelligence, I must admit that it is difficult to find much universities in the world who can provide complete and high-quality theoretical computer science modules like Aarhus University. Therefore, I would also like to thank all the teachers in the Department of Computer Science at Aarhus University who specialized in theoretical computer science.

I also want to thank my family. I have not been a hard-working student, but they still fully support me in pursuing what I like. I hope they can be healthy and happy.

Finally, I would also like to thank the Danish government and the administrative agencies of Aarhus University for providing many conveniences to international students from China. Under the volatile international situation and the increasingly tense relationship between China and the US, Denmark provides Chinese students with a safe, stable and cost-effective study destination. Especially during the epidemic, the Danish government also specially protected the right of international students to enter and exit Denmark. I hope that the epidemic will pass as soon as possible. I also hope that more and more countries can pursue equality and put people first, like the Nordic countries.

Bingsen Wang,
Baotou, Inner Mongolia, China, June 2021.

Contents

Chapter 1

Introduction

Digital signature is a fundamental cryptograhic primitive first introduced in 1976 [DH76]. It is a mathematical scheme for issuing and verifying the authenticity of digital messages or documents. Later, as more complex requirements emerged, such as hiding the signing message, hiding the identity of signer, split the sign ability to multiple server, more types of signature schemes were invented.
Like the ring signature proposed by Shamir in [RST01], which could perfectly hide the identity of the actual signer. The group signature proposed in [CvH91], which has a manager for identity of all the signer. Blind signature first proposed in [Cha83], which let the signer sign on the committed message, thus, the signer can not learn any information of the message. With secret sharing scheme proposed in [Sha79], secret key can be split to multiple server in which each server has zero information on the secret key but can generate a partial decryption or partial signature. With enough number of partial decryption or partial signature, it is easy to construct the plaintext or valid signature. This is called the threshold cryptosystem in general.
Following the development of pairing-based cryptography, several blind signature scheme was proposed, the Camenisch-Lysyanskaya (CL) signature proposed in [CL04] was widely used. But the size of CL signature grows with the number of message signed. In 2016, Pointcheval and Sanders (PS) proposed a short randomizable blind signature in [PS16], which has constant signature size with unlimited signing messages. Our work here, is to construct threshold blind signature scheme based on the signature scheme in [PS16]. To construct threshold cryptosystem, we always need secret sharing scheme. The Shamir's secret sharing scheme in [Sha79] was optimal but complicated to deploy and the share can not be the input to PRF to generate share of new random secret. On the contrary, replicated share is easy to deploy and can be the input to PRF to generate infinite number of share of new random secret. But replicated share takes too much space and time. A solution to this is to use share conversion proposed in [CDI05] to locally convert replicated share into Shamir's share.

Related Work Group signature scheme based on PS signature can be found in [CDL+20]. More threshold blind signature can be found in [KM15, VZK03, ADEO20].

Notation Our notation majorly follows the notation in [PS16, CDI05]. $\approx_k$ means the difference of the two operands exponentially converges to 0 with growth of parameter

k. PPT algorithm means probability polynomial time algorithm. In general, we say efficiently compute means it can be computed by PPT algorithm. $a \xleftarrow{\$} A$ means randomly select an element a from set A. $[n] = \{1,2,3,...,n\}$. We denote $\mathbb{G}_i^* = \mathbb{G}_i \backslash \{1_{\mathbb{G}_i}\}$.

Roadmap In chapter 2, we give our formal definition of threshold blind signature scheme and briefly introduce the necessary cryptographic primitive to construct our threshold blind signature scheme. Readers who are familiar with these knowledge can quickly skip it and return to it when needed. In chapter 3, we present our threshold blind signature scheme based on PS signature scheme, we present 3 different protocols based on different secret sharing scheme, we analyse their security and efficiency. In chapter 4, we implement our protocol on local machine based on BLS12-381 curve, and test for performance. In chapter 5, we discuss several application scenario of threshold blind signature scheme. In chapter 6, we conclude this thesis.

Tools This paper is generated with MacTex, our implementation is based on *blst* which is a implementation of BLS12-381 curve and BLS signature scheme. We use Go 1.16.3 darwin/amd64 to implement and test our proposed scheme.

Chapter 2

Preliminaries

In this chapter, we present several necessary preliminaries, from formal definition of Blind Digital Signature to some crypto primitives behind it like Commitment Scheme and Zero-knowledge Proof.

2.1 Blind Digital Signature

Following the notation in [DGK+20], we first present the definition of blind digital signature scheme.

Definition 1 *(Blind Signature Schemes). An interactive blind signature scheme between a signer S and user U consists of a tuple of efficient (probability polynomial time) algorithms* $\mathsf{BS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}_1, \mathsf{Sign}_2, \mathsf{Unblind}, \mathsf{VerifySig})$ *over message space M where*

- $\mathsf{Setup}(1^k)$*: On input the security parameter* 1^k*, output* pp*, which is implicitly given as input to all other algorithms.*
- $\mathsf{KeyGen}(\mathsf{pp})$*: On input the public parameter* pp *generates a key pair* $(\mathsf{pk}, \mathsf{sk})$ *with security parameter k.*
- $\mathsf{Sign}_1(\mathsf{pk}, m)$*: Run by U, takes as input* pk *and a message* $m \in M$ *and outputs* C *and* ω*.* C *can be thought of as the commitment on message m, U may be asked to prove the knowledge of m inside* C*.* ω *is the randomness used in this algorithm, which is necessary for unblinding the signature or opening the commitment* C*.*
- $\mathsf{Sign}_2(\mathsf{sk}, \mathsf{C})$*: Run by S, takes as input the commitment* C*. S may ask U to prove the knowledge of message m inside* C*, if U fails to prove, this algorithm aborts. Output the signature* σ'*.*
- $\mathsf{Unblind}(\sigma', \omega)$*:Run by U, with the randomness* ω *used in corresponding* Sign_1*, output the final signature* σ *which should be a valid signature for m under the public key* pk*.*
- $\mathsf{VerifySig}(\mathsf{pk}, m, \sigma)$*: Output a bit, which means accept or reject.*

Other than the unforgeablity, the security concern for the above blind signature scheme majorly comes from two aspects.

- For Sign_1, we want C has no information of m, thus, the signer can not learn any information of m.
- for Sign_2, we want σ' has no more information than the final signature σ, thus, user can not learn any information he is not supposed to learn.

Definition 2 *(Security of* BS*). An interactive blind signature scheme* $\mathsf{BS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}_1, \mathsf{Sign}_2, \mathsf{Unblind}, \mathsf{VerifySig})$ *is secure if the following three properties holds ($\approx_k$ here means the difference between two operands is negligible in parameter k):*

- *Existential Unforgeability under Chosen Message Attacks (Unforgeability): It is hard for any PPT adversary $\mathscr{A}$, even given access to a signing oracle, to output a valid signature σ for a message m which $\mathscr{A}$ never asked to the signing oracle.*
- *Blindness. For any PPT adversary $\mathscr{A}$,* $\Pr[\mathsf{Exp}^{\mathsf{Blind}}_{\mathscr{A}}(k) = 1] \approx_k 1/2$*, where the experiment* $\mathsf{Exp}^{\mathsf{Blind}}_{\mathscr{A}}(k)$ *is defined below.*
- *Simulatability. There exists a PPT algorithm* Sim *s.t. for any PPT adversary $\mathscr{A}$,* $\Pr[\mathsf{Exp}^{\mathsf{Sim}}_{\mathscr{A}}(k) = 1] \approx_k 1/2$*, where the experiment* $\mathsf{Exp}^{\mathsf{Sim}}_{\mathscr{A}}(k)$ *is defined below.*

Intuitively, *Blindness* says that the signer can not learn any non-negligible information of the message m and any adversary can not link any signature to any execution of the protocol. *Simulatability* says that the output of Sign_2 has no more information than the final signature which is supposed learn by the user.

$\mathsf{Exp}^{\mathsf{Blind}}_{\mathscr{A}}(k)$

1. $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^k)$; $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{pp})$;
2. $b \xleftarrow{\$} \{0,1\}$;
3. $(m_0, m_1) \leftarrow \mathscr{A}(\mathsf{pp}, \mathsf{sk})$;
4. $(\mathsf{C}, \omega) \xleftarrow{\$} \mathsf{Sign}_1(\mathsf{pk}, m_b)$; $(\overline{\mathsf{C}}, \overline{\omega}) \xleftarrow{\$} \mathsf{Sign}_1(\mathsf{pk}, m_{1-b})$;
5. $\sigma', \overline{\sigma'} \leftarrow \mathscr{A}(\mathsf{C}, \overline{\mathsf{C}})$;
6. $\sigma_b, \sigma_{1-b} \leftarrow \mathsf{Unblind}(\sigma', \omega), \mathsf{Unblind}(\overline{\sigma'}, \overline{\omega})$;
7. **if** $\mathsf{VerifySig}(\mathsf{pk}, m_0, \sigma_0) = 0 \vee \mathsf{VerifySig}(\mathsf{pk}, m_1, \sigma_1) = 0$ **then** $(\sigma_0, \sigma_1) = (\perp, \perp)$;
8. $b^* \leftarrow \mathscr{A}(\sigma_0, \sigma_1)$;
9. **if** $b = b^* \wedge |m_0| = |m_1|$, **then return** 1; **else return** 0;

$\mathsf{Exp}^{\mathsf{Sim}}_{\mathscr{A}}(k)$

1. $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^k)$; $(\mathsf{pk},\mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{pp})$;
2. $b \xleftarrow{\$} \{0,1\}$;
3. $(m;r) \leftarrow \mathscr{A}(\mathsf{pp},\mathsf{sk})$;
4. $(\mathsf{C},\omega) \xleftarrow{\$} \mathsf{Sign}_1(\mathsf{pk},m;r)$;
5. $\sigma'_0 \xleftarrow{\$} \mathsf{Sign}_2(\mathsf{sk},\mathsf{C})$;
6. $\sigma \leftarrow \mathsf{Unblind}(\sigma'_0,\omega)$;
7. $\sigma'_1 \xleftarrow{\$} \mathsf{Sim}(\mathsf{pk},m,\omega,\sigma)$;
8. $b^* \leftarrow \mathscr{A}(\sigma,m,\sigma'_b)$;
9. **if** $b = b^*$, **then return** 1; **else return** 0;

2.2 Threshold Blind Signature Scheme

The concept of threshold cryptosystem was first introduced by Shamir in [Sha79]. With secret-sharing scheme, the secret key can be shared by n servers such that any subset of more than t servers can reconstruct the secret key (any subset of size no more than t has zero information on the secret), we call this as (n,t)-threshold cryptosystem. In such a system the encryption or sign procedure can be split to $t+1$ part with each chosen server computes one part and the user can construct the ciphertext or signature which is valid for the corresponding public key. Following the definition of threshold encryption scheme in [DGK+20] , we give a definition of threshold blind signature scheme as below.

Definition 3 *(Threshold Blind Signature Schemes). A* (n,t)*-threshold blind signature scheme* $\mathsf{TBS} = (\mathsf{Setup},\mathsf{TKeyGen},\mathsf{Sign}_1,\mathsf{ShareSign}_2,\mathsf{TCombine},\mathsf{TVerifySig})$ *over message space M with* $n \geq 2t+1$ *consists of the following PPT algorithms:*

- $\mathsf{Setup}(1^k)$*: On input the security parameter* 1^k*, output* pp*, which is implicitly given as input to all other algorithms.*
- $\mathsf{TKeyGen}(1^k)$*: Takes the public parameter* pp *as input and randomly returns a public-private key pair* $(\mathsf{pk},\mathsf{sk})$*. The private key* sk *should be shared by n servers* $\{P_1,...,P_n\}$ *with a* (n,t)*-threshold secret sharing scheme such that server* P_i *holds share* sk_i*.*
- $\mathsf{Sign}_1(\mathsf{pk},m)$*: Run by U, takes as input* pk *and a message* $m \in M$ *and computes* C *and* ω*.* C *can be thought of as the commitment on message m. U may be asked to prove the knowledge of m inside* C*.* ω *is the randomness used for the commitment* C *in this algorithm, which is necessary for unblinding the signature or opening*

the commitment C*. Randomly choose a subset* $I \subseteq [n]$ *with* $|I| = t+1$*, sends* C *to* P_i *with* $i \in I$*.*

- $\mathsf{ShareSign}_2(\mathsf{sk}_i, \mathsf{C})$*: On input the commitment* C *and share of secret key* sk_i*, player* P_i *may ask* U *to prove the knowledge of message inside* C*, if satisfied, player* P_i *sends to* U *the partial signature as* μ_i*.*

- $\mathsf{TCombine}(\{\mu_i\}_{i\in I}, \omega)$*: An algorithm that takes the subset* $I \subseteq [n]$ *with size* $t+1$*, randomness* ω *chosen in* Sign_1 *and collection of partial signatures* $\{\mu_i\}_{i\in I}$*, construct the final signature* σ*.*

- $\mathsf{TVerifySig}(\mathsf{pk}, m, \sigma)$*: Output* 1 *if* σ *is a valid signature for* m *under public key* pk*, otherwise, output* 0*.*

Adversary The most powerful adversary $\mathscr{A}$ we can tolerate is the PPT adversary who can corrupts at most t servers and we allow $\mathscr{A}$ ask signatures for several messages. The security we want is for any such PPT $\mathscr{A}$, he has negligible probability to produce a valid signature for a message he never asked. To formally define this, we generalise the concept *Unforgeability* for threshold signature. We denote the index of t corrupted servers as C, $t+1$ chosen server in one signature as I, in worst case, we have

$$C = \{S_1, ..., S_t\} \subset I = \{S_1, ..., S_{t+1}\} \subseteq \{1, ..., n\}.$$

Other than the unforgeablity, the security concern for the above threshold blind signature scheme majorly comes from three aspects.

- Even if t servers are corrupted or halt, it will not make the the entire signature system fail.

- For Sign_1, we want C has no information of m, thus, any signer can not learn any non-negligible information of m.

- for $\mathsf{ShareSign}_2$, we want all the partial signature $\{\mu_i\}_{i\in I}$ has no more information than the final signature σ, thus, user won't learn anything he is not supposed to learn.

Definition 4 *(Security of* TBS*). An interactive threshold blind signature scheme* $\mathsf{TBS} = (\mathsf{Setup}, \mathsf{TKeyGen}, \mathsf{Sign}_1, \mathsf{ShareSign}_2, \mathsf{TCombine}, \mathsf{TVerifySig})$ *is secure if the following four properties holds (*$\approx_k$ *here means the difference between two operands is negligible in parameter* k*):*

- *Existential Unforgeability under Chosen Message Attacks (Unforgeability): It is hard for any PPT adversary* $\mathscr{A}$*, even given access to a signing oracle, to output a valid signature* σ *for a message* m *which* $\mathscr{A}$ *never asked to the signing oracle.*

- *Robustness. Even* t *servers are corrupted, the user* U *still can efficiently get a valid signature.*

- *Static Semantic Security (SSS). For any PPT adversary* $\mathsf{Exp}^{\mathsf{SSS}}_{\mathscr{A}}(k, \mathsf{n}, \mathsf{t}) = 1] \approx_k 1/2$*, where the experiment* $\mathsf{Exp}^{\mathsf{SSS}}_{\mathscr{A}}(k, \mathsf{n}, \mathsf{t})$ *is defined below.*

- *Partial Signature Simulatability (PSS). There exists a PPT algorithm* SimPart *s.t. for any PPT adversary* $\mathscr{A}$*,* $\Pr[\mathsf{Exp}^{\mathsf{PSS}}_{\mathscr{A}}(k,\mathsf{n},\mathsf{t})=1] \approx_k 1/2$*, where the experiment* $\mathsf{Exp}^{\mathsf{PSS}}_{\mathscr{A}}(k,\mathsf{n},\mathsf{t})$ *is defined below.*

$\mathsf{Exp}^{\mathsf{SSS}}_{\mathscr{A}}(k,\mathsf{n},\mathsf{t})$

1. $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^k)$; $(\mathsf{pk},\{\mathsf{sk}_i\}_{i\in[n]}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{pp})$;
2. $b \xleftarrow{\$} \{0,1\}$
3. $(I,m_0,m_1) \leftarrow \mathscr{A}(\mathsf{pk},\{\mathsf{sk}_i\}_{i\in[n]})$; in which $(I \subseteq [n] \wedge |I| = t+1)$.
4. $(\mathsf{C},\omega) = \mathsf{Sign}_1(\mathsf{pk},m_b)$; $(\overline{\mathsf{C}},\overline{\omega}) = \mathsf{Sign}_1(\mathsf{pk},m_{1-b})$;
5. $(\{\mu_i\}_{i\in I},\{\overline{\mu}_i\}_{i\in I}) \leftarrow \mathscr{A}(\mathsf{C},\overline{\mathsf{C}})$;
6. $\sigma_b,\sigma_{1-b} \leftarrow \mathsf{TCombine}(\{\mu_i\}_{i\in I},\omega),\mathsf{TCombine}(\{\overline{\mu}_i\}_{i\in I},\overline{\omega})$
7. **if** $\mathsf{TVerifySig}(\mathsf{pk},m_0,\sigma_0) = 0 \vee \mathsf{TVerifySig}(\mathsf{pk},m_1,\sigma_1) = 0$ **then** $(\sigma_0,\sigma_1) = (\perp,\perp)$;
8. $b^* \leftarrow \mathscr{A}(\sigma_0,\sigma_1)$;
9. **if** $b = b^* \wedge |m_0| = |m_1|$, **then return** 1; **else return** 0;

$\mathsf{Exp}^{\mathsf{PSS}}_{\mathscr{A}}(k,\mathsf{n},\mathsf{t})$

1. $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^k)$; $(\mathsf{pk},\{\mathsf{sk}_i\}_{i\in[n]}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{pp})$;
2. $C \leftarrow \mathscr{A}(\mathsf{pp},k,\mathsf{n},\mathsf{t})$;in which $(C \subset [n] \wedge |C| \leq t)$.
3. $(I,m,r) \leftarrow \mathscr{A}(\mathsf{pk},\{\mathsf{sk}_i\}_{i\in[n]})$; in which $(I \subseteq [n] \wedge |I| = t+1)$.
4. $b \xleftarrow{\$} \{0,1\}$;
5. $(\mathsf{C},\omega) \xleftarrow{\$} \mathsf{Sign}_1(\mathsf{pk},m;r)$;
6. $\mu_i^0 = \mathsf{ShareSign}_2(\mathsf{sk}_i,\mathsf{C})$ **for** $i \in I$; $\sigma = \mathsf{TCombine}(\{\mu_i^0\}_{i\in I})$;
7. $\mu_i^1 = \mu_i^0$ **for** $i \in I \cap C$;
8. $\{\mu_i^1\}_{i\in I\setminus C} = \mathsf{SimPart}(\mathsf{n},\mathsf{t},\mathsf{C},\sigma,\omega,\{\mu_j^0\}_{j\in I\cap C})$;
9. $b^* \leftarrow \mathscr{A}(m,\sigma,\{\mu_j^b\}_{j\in I\setminus C})$;
10. **if** $b = b^*$, **then return** 1; **else return** 0;

2.3 Bilinear Group

In this section we present the definition of bilinear group which is used in this paper. Bilinear groups are a tuple of three cyclic groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$, both of prime order p along with a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties:

- for any $g \in \mathbb{G}_1, \tilde{g} \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$, we always have $e(g^a, \tilde{g}^b) = e(g, \tilde{g})^{a \cdot b}$.
- for $g \neq 1_{\mathbb{G}_1}, \tilde{g} \neq 1_{\mathbb{G}_2}$, we always have $e(g, \tilde{g}) \neq 1_{\mathbb{G}_T}$.
- the map e is efficiently computable.

In [GPS08], Galbraith, Paterson, and Smart defined three types of pairing by giving more constraints.

1. in type 1 pairing, $\mathbb{G}_1 = \mathbb{G}_2$.
2. in type 2 pairing, $\mathbb{G}_1 \neq \mathbb{G}_2$ while there exists an efficiently computable homomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$. Note that this ϕ will make DDH problem in $\mathbb{G}_2$ easy but remain hard in $\mathbb{G}_1$.
3. in type 3 pairing, $\mathbb{G}_1 \neq \mathbb{G}_2$ and no efficiently computable homomorphism exists between $\mathbb{G}_1$ and $\mathbb{G}_2$, in either direction.

In this paper, as in [PS16], our proposed scheme is based on the type 3 pairing. If the DDH problem is believed to be hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$, we can believe that there is no efficiently computable homomorphism between them.
By choosing certain groups $\mathbb{G}_1$ and $\mathbb{G}_2$ carefully, we can get better efficiency and the corresponding assumption is called symmetric XDH assumption which we will present below.

2.4 Assumptions

The security of cryptosystem is always reduced to some computational problem believed hard for now. In other words, in order to reason the security of cryptosystem, we need some assumptions like some problem is not efficiently solvable. In this section, we present the assumptions we need in this paper.

Definition 5 *(Decisional Diffie-Hellman problem (DDH)): Let $\mathbb{G}$ be a cyclic group of prime order p written additively. The DDH problem is to distinguish the two distributions in $\mathbb{G}^4$:*

$$\{(P, aP, bP, abP) : P \in G, 0 \leq a, b < p\}$$

and

$$\{(P, aP, bP, cP) : P \in G, 0 \leq a, b, c < p\}.$$

Likewisely, given aP to compute a is called Discrete Logarithm Problem (DLP). Given aP, bP to compute abP is called Computational Diffie-Hellman problem (CDH). It's easy to see that DDH assumption is stronger than CDH assumption and DLP assumption.

Definition 6 *(Generalised Decisional Diffie-Hellman problem (co-DDH)): Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic group of prime order p. The co-DDH problem is to distinguish the two distributions in in $\mathbb{G}_1^2 \times \mathbb{G}_2^2$:*

$$\{(P, aP, Q, aQ) : P \in \mathbb{G}_1, Q \in \mathbb{G}_2, 0 \leq a < p\}$$

and

$$\{(P, aP, Q, cQ) : P \in \mathbb{G}_1, Q \in \mathbb{G}_2, 0 \leq a, c < p\}.$$

To be more precise, when we say some problem is hard in some group, which means there is no probability polynomial time (PPT) algorithm which can solve the problem in the group. Now, we can present the assumption for type 3 pairing, which is the basis of PS signature and our proposed scheme.

Definition 7 *(symmetric external Diffie-Hellman (SXDH) assumption): There exists two distinct groups $\mathbb{G}_1, \mathbb{G}_2$ such that :*

- *DLP, CDH, co-CDH are both hard in $\mathbb{G}_1, \mathbb{G}_2$.*
- *There exists an efficiently computable bilinear map (pairing) $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.*
- *DDH is hard in $\mathbb{G}_1, \mathbb{G}_2$.*

In the implementation part, we can see that for now, the most efficient way to construct pairing with SXDH assumption is by using BLS curve proposed in [BLS03]. By choosing certain subgroup of ordinary elliptic curve, we can believe that DDH is hard. Specific reason about this can be found in [GR04]. More pairing friendly curve can be found in [BN06].

Definition 8 *(LRSW assumption): Let $\mathbb{G}$ be a cyclic group of prime order p, with a generator g. For $(X, Y) = (g^x, g^y)$, where $(x, y) \xleftarrow{\$} \mathbb{Z}_p^2$, we define the oracle $\mathcal{O}(m)$ on input $m \in \mathbb{Z}_p$ that take $h \xleftarrow{\$} \mathbb{G}$ and outputs a triple $T = (h, h^y, h^{x+mxy})$. Given (X, Y) and unlimited access to this oracle, no PPT adversary can generate such a valid triple for a never asked $m^* \in \mathbb{Z}_p$.*

This assumption was introduced and proved in [LRSW00]. Two similar assumptions as below were proposed in [PS16] and is the basis of the unforgeability of the PS signature scheme.

Definition 9 *(PS assumption 1): Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, e)$ be a type 3 pairing. With $g, \tilde{g}$ be the generator of $\mathbb{G}_1, \mathbb{G}_2$. For $(X, Y, \tilde{X}, \tilde{Y}) = (g^x, g^y, \tilde{g}^x, \tilde{g}^y)$ where $(x, y) \xleftarrow{\$} \mathbb{Z}_p^2$, we define the oracle $\mathcal{O}(m)$ on input $m \in \mathbb{Z}_p$ that outputs a pair $P = (h, h^{x+ym})$, where $h \xleftarrow{\$} \mathbb{G}_1$. Given $(g, \tilde{g}, Y, \tilde{X}, \tilde{Y})$ and unlimited access to this oracle, no PPT adversary can generate such a valid pair with $h \neq 1_{\mathbb{G}_1}$ for a never asked $m^* \in \mathbb{Z}_p$.*

Definition 10 *(PS assumption 2): Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, e)$ be a type 3 pairing. With $g, \tilde{g}$ be the generator of $\mathbb{G}_1, \mathbb{G}_2$. For $(\tilde{X}, \tilde{Y}) = (\tilde{g}^x, \tilde{g}^y)$ where $(x, y) \xleftarrow{\$} \mathbb{Z}_p^2$, we define the oracle $\mathcal{O}(m)$ on input $m \in \mathbb{Z}_p$ that outputs a pair $P = (h, h^{x+ym})$, where $h \xleftarrow{\$} \mathbb{G}_1$. Given $(\tilde{g}, \tilde{X}, \tilde{Y})$ and unlimited access to this oracle, no PPT adversary can generate such a valid pair with $h \neq 1_{\mathbb{G}_1}$ for a never asked $m^* \in \mathbb{Z}_p$.*

The above two assumptions holds in generic bilinear group model, the proof can be found in the appendix of [PS16].

Theorem 1 *PS assumption 1 and PS assumption 2 both holds in the generic bilinear group model: after q oracle queries and w group-oracle queries, no adversary can generate a valid pair for a never asked $m^* \in \mathbb{Z}_p$ with probability greater than $6(q+w)^2/p$.*

2.5 Pointcheval-Sanders Signature Scheme

In [PS16], an short randomizable blind signature scheme was proposed. It supports message aggregate, with zero-knowledge proof, it can be an efficient solution for anonymous credential problem proposed in [CL04]. In this section, we first see how this scheme works on single message and committed message which makes it a blind signature scheme.

2.5.1 Signature on Single Message

PS signature on single message

- $\mathsf{Setup}(1^k)$: Given a security parameter k, this algorithm outputs $pp \leftarrow (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$. The bilinear group must be of type 3.
- $\mathsf{Keygen}(pp)$: Selects $\tilde{g} \xleftarrow{\$} \mathbb{G}_2^*$ and $(x,y) \xleftarrow{\$} \mathbb{Z}_p^2$, computes $(\tilde{X}, \tilde{Y}) \leftarrow (\tilde{g}^x, \tilde{g}^y)$. Set $\mathsf{sk} \leftarrow (x,y)$ and $\mathsf{pk} \leftarrow (\tilde{g}, \tilde{X}, \tilde{Y})$.
- $\mathsf{Sign}(\mathsf{sk}, m)$: Selects a random $h \xleftarrow{\$} \mathbb{G}_1^*$ and outputs signature $\sigma \leftarrow (h, h^{(x+y\cdot m)})$.
- $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$: Parse σ as (σ_1, σ_2), check if $\sigma_1 \neq 1_{\mathbb{G}_1}$ and $e(\sigma_1, \tilde{X} \cdot \tilde{Y}^m) = e(\sigma_2, \tilde{g})$ then accept, otherwise reject.

Correctness: If $\sigma = (\sigma_1 = h, \sigma_2 = h^{(x+y\cdot m)})$, then we have

$$e(\sigma_1, \tilde{X} \cdot \tilde{Y}^m) = e(h, \tilde{g})^{(x+y\cdot m)} = e(h^{(x+y\cdot m)}, \tilde{g}) = e(\sigma_2, \tilde{g}).$$

Security: The security of PS signature scheme is based on the bilinear group satisfying the SXDH assumption. Specifically, the unforgeability of PS signature on single message can be reduced to PS assumption 2.

Randomizability: Given a valid signature $\sigma = (\sigma_1, \sigma_2)$ on a message m. With $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$, $\sigma' = (\sigma_1^\gamma, \sigma_2^\gamma)$ is still a valid signature on m. It corresponds to replace $h \in \mathbb{G}_1^*$ by $h' = h^\gamma \in \mathbb{G}_1^*$.

2.5.2 Signature on Committed Message

With some modification on basic PS signature scheme and zero-knowledge proof, Pointcheval and Sanders also proposed a blind signature scheme as below.

Blind PS signature on single message

- $\mathsf{Setup}(1^k)$: Given a security parameter k, this algorithm outputs $pp \leftarrow (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$. The bilinear group must be of type 3. We denote $\mathbb{G}_i^* = \mathbb{G}_i \backslash \{1_{\mathbb{G}_i}\}$.
- $\mathsf{Keygen}(pp)$: Selects $(g, \tilde{g}) \xleftarrow{\$} (\mathbb{G}_1^*, \mathbb{G}_2^*)$ and $(x, y) \xleftarrow{\$} \mathbb{Z}_p^2$, computes $(X, Y) \leftarrow (g^x, g^y), (\tilde{X}, \tilde{Y}) \leftarrow (\tilde{g}^x, \tilde{g}^y)$. Set $\mathsf{sk} \leftarrow X$ and $\mathsf{pk} \leftarrow (g, Y, \tilde{g}, \tilde{X}, \tilde{Y})$.
- $\mathsf{Sign}_1(m)$: With a message $m \in \mathbb{Z}_p$, the user select random $\omega \xleftarrow{\$} \mathbb{Z}_p$ and sends the commitment $\mathsf{C} \leftarrow g^{\omega} Y^m$ to the signer. Store ω for future unblinding.
- $\mathsf{Sign}_2(\mathsf{sk}, \mathsf{C})$: The signer first ask user to proof the knowledge of the message inside the commitment C. If the signer is convinced, he selects $u \xleftarrow{\$} \mathbb{Z}_p$ and returns $\sigma' \leftarrow (g^u, (X\mathsf{C})^u)$.
- $\mathsf{Unblind}(\sigma')$: The user unblind the signature by computing $\sigma \leftarrow (\sigma'_1, \sigma'_2 / \sigma'^{\omega}_1)$. Where ω was stored in Sign_1.

The above protocol can generate a valid PS signature on a committed message m. To prove the knowledge of the signature without revealing m, the user still needs zero-knowledge proof.
Similarly, the unforgeability of the blind PS signature scheme can be reduced to PS assumption 1.
With little modification, we can present the PS signature on committed multiple message as below.

Blind PS signature on multiple message

- $\mathsf{Setup}(1^k)$: Given a security parameter k, this algorithm outputs $pp \leftarrow (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$. The bilinear group must be of type 3.
- $\mathsf{Keygen}(pp)$: Selects $(g, \tilde{g}) \xleftarrow{\$} (\mathbb{G}_1^*, \mathbb{G}_2^*)$ and $(x, y_1, y_2, ..., y_r) \xleftarrow{\$} \mathbb{Z}_p^{r+1}$, computes $(X, Y_1, Y_2, ..., Y_r) \leftarrow (g^x, g^{y_1}, g^{y_2}, ..., g^{y_r}), (\tilde{X}, \tilde{Y}_1, \tilde{Y}_2, ..., \tilde{Y}_r) \leftarrow (\tilde{g}^x, \tilde{g}^{y_1}, \tilde{g}^{y_2}, ..., \tilde{g}^{y_r})$. Set $\mathsf{sk} \leftarrow X$ and $\mathsf{pk} \leftarrow (g, \{Y_i\}_{i\in[r]}, \tilde{g}, \tilde{X}, \{\tilde{Y}_i\}_{i\in[r]})$.
- $\mathsf{Sign}_1(\{m_i\}_{i\in[r]})$: With a set of message $m_i \in \mathbb{Z}_p$ for $i \in [r]$, the user selects $\omega \xleftarrow{\$} \mathbb{Z}_p$ and sends the commitment $\mathsf{C} \leftarrow g^{\omega} \prod_{i\in[r]} Y_i^{m_i}$ to the signer. Store ω for future unblinding.
- $\mathsf{Sign}_2(\mathsf{sk}, \mathsf{C})$: The signer first ask user to proof the knowledge of the messages inside the commitment C. If the signer is convinced, he selects $u \xleftarrow{\$} \mathbb{Z}_p$ and returns $\sigma' \leftarrow (g^u, (X\mathsf{C})^u)$.
- $\mathsf{Unblind}(\sigma')$: The user unblind the signature by computing $\sigma \leftarrow (\sigma'_1, \sigma'_2/\sigma'^{\omega}_1)$. Where ω was stored in Sign_1.

The final signature σ can be verified by checking

$$e(\sigma_1, \tilde{X} \cdot \prod_{i\in[r]} \tilde{Y}_i^{m_i}) = e(\sigma_2, \tilde{g})$$

. Obviously, it can be randomized similarly as PS signature on single message.

Security: The unforgeability of the above signature scheme can be reduced to PS assumption 1, complete proof can be found in the appendix of [PS16]. Their prove strategy is by constructing an adversary who can break PS assumption 1 from an adversary who can break unforgeability of the blind PS signature scheme on multiple message.

2.6 Secret Sharing Scheme

Following the notation in [CDI05], we define secret sharing scheme for n-player $\{P_1, ..., P_n\}$ by a tuple $\mathscr{S} = (K, (S_1, ..., S_n), R, D)$, where K is a finite *secret-domain*, each S_j is a finite *share-domain* denoting P_j's share, typically $S_j = K^{a_j}$ for some a_j, R is a probability distribution from which the dealer's randomness picked, and D is a share distribution function mapping a secret s and a random input r to an n-tuple of shares in $S_1 \times \cdots \times S_n$.

Access Structure. If a subset T of players can reconstruct the secret s by combining their shares, we say T is an qualified set, otherwise, T is an unqualified set. We define *access structure* $\mathscr{T}$ as the collection of all qualified sets and $\overline{\mathscr{T}}$ as the collection of all the maximal unqualified sets.

(n,t)**-Threshold secret-sharing.** To construct threshold cryptosystem, one useful type of secret-sharing scheme is *threshold schemes* in which the access structure is the set of all the subset of players with size more than some threshold t. It's straightforward to see that in an (n,t)-threshold secret-sharing scheme, the number of different maximal unqualified set of player is $\binom{n}{t}$ which means

$$|\overline{\mathscr{T}}| = \binom{n}{t}$$

.

2.6.1 Replicated Secret Sharing

To share a secret $s \in K$ to n players $\{P_1, ..., P_n\}$, we can randomly split s as $s = \sum_{i\in[n]} r_i$ and distribute r_i to P_i. In this case, unless an adversary has all the shares, he has no information on the secret s. This scheme is safe and simple but doesn't fit our application since there is only one qualified set which is all the players. For any access structure $\mathscr{T}$, we can give a general replicated secret sharing scheme as below.

threshold Replicated secret-sharing scheme

- Common input: threshold (n,t). Access structure $\mathscr{T}$ includes all the qualified sets.
- Input to dealer: Secret $s \in K$.
  1. From $\mathscr{T}$, the dealer computes the collection of all the maximal unqualified sets as $\overline{\mathscr{T}}$.
  2. Split s as $s = \sum_{T\in\overline{\mathscr{T}}} r_T$.
  3. Distribute r_T to player P_j if $j \notin T$.

In the above scheme, to make sure any unqualified set has no information on the secret, the dealer create a replicated variable for each maximal unqualified set. This will work but not efficiently.

2.6.2 Shamir's Secret Sharing

To achieve more efficiency, Shamir's Secret Sharing would be our choice. It was first proposed by Adi Shamir in [Sha79]. From Lagrange interpolation, we know that we can determine a degree t polynomial over finite fields if we have $t+1$ different sample points. With this idea in mind, we can understand Shamir's Secret Sharing scheme as below.

Shamir's Secret Sharing scheme

- Common input: threshold (n,t).
- Input to dealer: Secret $s \in K$.
  1. The dealer randomly choose a degree t polynomial f over K with $f(0) = s$.
  2. Distribute $f(i)$ to player P_i as his share.

This is much more efficient than replicated secret sharing since each player only have to hold one share with same size as the secret.
To explain some useful property of Shamir's secret sharing, with degree t polynomial f, $f(0) = s$, $n > t$, we give a short notation of sharings as

$$[s;f]_t = \{f(1), f(2), ..., f(n)\}$$

For another secret a, we have $[a;g]_t = \{g(1), g(2), ..., g(n)\}$. Then we have

$$[s+a;f+g]_t = \{f(1)+g(1), f(2)+g(2), ..., f(n)+g(n)\}$$

With $n \geq 2t+1$, we can also compute the share for secret sa similarly.

$$[sa;fg]_{2t} = \{f(1)g(1), f(2)g(2), ..., f(n)g(n)\}$$

This property is called *homomorphic*. With this property, each player can compute the share for a new secret locally.

2.6.3 Pseudorandom Secret-Sharing(PRSS)

For n players $\{P_1, ..., P_n\}$ using replicated share as in 2.6.1 hold share on a secret s, if they already agreed on some other value a, they can further locally generate share on random secret with PRF ψ.

PRSS

- Common input: All the n players $\{P_1, ..., P_n\}$ has agreed on a value a and replicated share r_T distributed as in 2.6.1.
- Player P_j locally compute $r'_T = \psi_{r_T}(a)$ for all $j \notin T$.

In our proposed scheme, both player have to keep updating their share for new secret key and random value. With PRSS, there is no need for communication between servers for updating share.

2.6.4 Share Conversion

In [CDI05], they proposed a general method to locally convert replicated share into share of any linear secret sharing scheme. Specifically, we can first distribute replicated share for the secret key to all the server, and the server can later convert their share into Shamir's share to save space. They proved that $\binom{n}{t}$ overhead is necessary in this kind of

conversion, and the conversion on reverse direction is impossible.
We present how to convert replicated share into Shamir's share locally.

Convert replicate share to Shamir's share locally

For any A let f_A be the unique $degree-t$ polynomial such that:

- $f_A(0) = 1$
- $f_A(i) = 0$ for all $i \in A$.(This is necessary since P_i doesn't know r_A for $i \in A$)

Each player P_j computes a share s_j as follows:

$$s_j = \sum_{A \subseteq [n]:|A|=t, j \notin A} r_A \cdot f_A(j)$$

Define $degree-t$ polynomial $f = \sum_A r_A \cdot f_A$.
It's easy to verify that s_j is a valid Shamir's share for f and $f(0) = s$.
We can replace r_A with $\psi_{r_A}(a)$ to further generate more Shamir's share for random value.

In our proposed scheme, for each threshold signature, all the servers need to prepare shares for a new random secret. With share conversion and PRSS, we can first let the trusted dealer to distribute randomness and replicated share to servers. As the replicated share is uniformly random and independent with each other when the secret is random, it can be used as key to the PRF to generate share for a new random secret. The server can generate replicated share for infinite random secret, to store them, server can convert them into Shamir's share locally. In this way, server can prepare enough share off-line, when it is asked to sign something on-line, server can use the already prepared Shamir's share.
Therefore, share conversion allows us to enjoy the best of both worlds, combining the share independence advantage of the replicated share with the space-optimal advantage of Shamir's share.

2.7 Commitment Scheme

Commitment schemes are necessary in our definition of blind signature scheme. In this section, we briefly introduce the concept of commitment schemes and some important results. A commitment scheme is a tuple of two probabilistic polynomial algorithm $(\mathscr{G}, \mathsf{commit}_{pk})$. We define each algorithm below and present how the commitment schemes runs between two player P, V in whcih P wants to commit a message m and later open it to V.

Commitment Scheme

- $\mathscr{G} : 1^l \to pk$ is a public key generator which take input l as security parameter and output a public key pk.
- $\mathsf{commit}_{pk} : \{0,1\}^l \times \{0,1\}^* \to \{0,1\}^l$ takes a random bit string with length l and the message m to commit and output the commit as a bit string with length l.

  1. Either party P or V runs $\mathscr{G}$ and send pk to another party.
  2. P randomly choose $r \in \{0,1\}^l$ and compute $C \leftarrow \mathsf{commit}_{pk}(r,m)$. Send the commitment C to V.
  3. P opens the commitment C by sending (r,m) to V. V checks if $C = \mathsf{commit}_{pk}(r,m)$.

Based on RSA, discrete logarithm or secure hash function, it's easy to implement a commitment scheme as defined above. In order to analysis the security of commitment schemes, we have to consider from two aspects, namely *binding* and *hiding*.

- *binding*: Player P can not change the message inside a commitment C. Specifically, P can not find two tuple $(r,m),(r',m')$ with $m' \neq m$ such that $\mathsf{commit}_{pk}(r,m) = \mathsf{commit}_{pk}(r',m')$.
- *hiding*: Player V can not get non-negligible information on the message inside the commitment unless P open it.

When we say *unconditional binding* or *hiding*, it means even P or V has unlimited computation power, he can not break the corresponding property.
When we say *computational binding* or *hiding*, it means when P or V only has polynomial bounded computation power, he can not break the corresponding property.
It's easy to see that our definition can only achieve *unconditional hiding* and *computational binding* since the space of commitment is a proper subset of the space of (r,m). Actually, it's impossible to achieve *unconditional* secure on both *hiding* and *binding* property.
One may ask, is there exists commitment schemes with *unconditional binding* and *computational hiding*? An important result can be found in [Nao90].

Theorem 2 *If one-way functions exist, then commitment schemes with unconditional binding and computational hiding exist.*

2.8 Zero-knowledge Proof

In our definition of blind signature, after sending the commitment of message to the signer, the user U needs to prove that he knows the message inside the commitment. Of course U can send m to the signer, but this is equivalent to open the commitment. As we don't want the signer get any information on the message m, the only information U wants to leak to signer is that he knows m but not m. To formalise this, we need to introduce the concept of Zero-knowledge Proof.

Definition 11 *(ZKIP). A zero knowledge proof system is a interactive proof system with following 3 properties:*

- *Completeness: if the statement is true, the honest verifier will be convinced by an honest prover.*
- *Soundness: if the statement is false, no cheating prover can convince the honest verifier with non-negligible probability.*
- *Zero-knowledge: if the statement is true, no verifier learns any information other than the fact that the statement is true.*

Goldreich, Micali and Wigderson [GMW86] shows that any **NP** $\subset$ **ZKIP** if commitment schemes with unconditional binding exist. Combined with theorem 2, we know that if one way function exists, then **NP** $\subset$ **ZKIP**. Their proof strategy is to construct a zero-knowledge proof protocol for a **NP** complete language and reduce other language in **NP** to it. In other word, everything provable is provable in zero-knowledge [BOGG$^+$90]. This concept looks very abstract at first glance, let's see some simple example below to help you better understand. In the first example, the prover wants to prove that he knows the discrete log of a given value which is in a group where DL is supposed to be hard. In the second example, the prover wants to prove that he knows g^x where x is a secret hold by the verifier. We use P to denote the Prover, V to denote the Verifier.

Proving knowledge of Discrete log

- Input to P: Value x, generator g. (x should be a big random integer. g should be a generator of a group with large prime order and DL is supposed to be hard in it.)
- Input to V: Value $y = g^x$.
- Run a large enough number of rounds of protocol below.
  1. P randomly choose a big integer r and send $C = g^r$ to V.
  2. V randomly choose $b \in \{0,1\}$. If $b = 0$, V ask P to send r back, other wise V ask P to send $d = x + r$ back.
  3. P send r or $d = x + r$ back as asked by V.
  4. If $b = 0$, V checks if $C = g^r$, otherwise V checks if $g^d = Cy$. If the check fail, V reject, otherwise V choose to believe the statement that P knows the discrete log of y.

Let's try to prove the above protocol is a zero knowledge proof system. As honest Prover with the discrete log could always give the correct value in each step, thus, he can make an honest Verifier always be convinced. Thus, the above protocol has Completeness. As DL is hard, the cheating prover has only $1/2$ chance to make verifier convinced in each round, with n rounds, the probability become $1/2^n$ which is negligible if n is big enough. Thus, the above protocol has Soundness. The information received by V can be simulated by V locally, thus, V learns no more information than the final statement is true or false, thus, the above protocol has Zero-knowledge. Therefore, the above protocol is a zero knowledge proof system.

Proving knowledge of g^x

- Input to P: Value $y = g^x$.
- Input to V: Value x, generator g. (x should be a big random integer. g should be a generator of a group with large prime order and DL is supposed to be hard in it.)
- 1. P send y to V.
  2. V checks if $y = g^x$. If fail, V reject, otherwise, V believe that P knows g^x.

There is other more efficient way to do this, but the protocol above could be help in our scheme in 3.3.1.

Chapter 3

Our Proposed Schemes

In this chapter, we present 3 version of threshold PS signature scheme and talk about how to do the verification anonymously. Our goal is to build a threshold blind signature scheme without allowing communication between each server while allowing communication between server and a trusted dealer or user.

3.1 Design Principle

In this section, we briefly explain our design idea and principle in a relatively high level. We hope this will help you better understand our protocol in the following 3 sections. Generally speaking, we want to design a threshold blind signature scheme based on Pointcheval-Sanders signature scheme which was proposed in [PS16, PS17]. The version for multiple message is not much different from version of single message, for simplicity, we will only construct the TBS for multiple message in section 3.4.1.
Our protocol will follow the definition in 1. Besides, we want to forbid the communication between any two server and we will allow a trusted dealer to communicate with each server.

Trusted Dealer. A trusted dealer will strictly follow the protocol, will not be corrupted by any adversary and will not leak any information. Remember in 2.5.2, user give the commitment C of his message to signer, signer based on the secret key X and random value u to compute $(g^u, (X\mathsf{C})^u)$ and send it back. With secret sharing, we can split x and u, we let each server hold a share of x and u thus he can compute partial signature. After the user get the partial signature from chosen server, we want that user can efficiently verify if the partial signature is correctly generated thus our protocol will be robust. Then, user can construct the final signature.
We will analysis the security and efficiency of our scheme. For the security, we will follow the definition in 1, thus majorly four aspects, *Unforgeability*, *Robustness*, *Static Semantic Security*, *Partial Signature Simulatibility*.

Share of u Actually, we can let each server randomly select share of u locally. This is equivalent with choosing a different u. But this will be different from the PS signature scheme. As the signer knows the random value u chosen for each signature, we want any qualified set of server knows the random value u for each signature. Thus, we need

all the server has share of a random value u instead of choosing it randomly. And we want all the server update the share for a new random u after finishing each signature.

3.2 TBS based on Replicated Secret-Sharing

In this section, we present the threshold PS signature scheme based on replicated secret share. This is a relatively inefficient protocol, but easy to understand. In the following section, we will present more efficient solution.

3.2.1 The Protocol

With replicated secret-sharing and pseudorandom secret-sharing, all the n servers $\{P_1, ..., P_n\}$ have replicate share on $\mathsf{sk} : x$ and several random u. To get a valid blind PS signature as in 2.5.2, user needs to reconstruct g^u and $g^{x \cdot u}$ with partial signature returned from the $t+1$ server.

Threshold PS signature based on Replicated secret-share

- Invariant: $x = \sum_{T \in \overline{\mathscr{T}}} r_T^x$, $K = \sum_{T \in \overline{\mathscr{T}}} r_T^K$, $\tilde{X}_T = \tilde{g}^{r_T^x}$. $\overline{\mathscr{T}}$ is the collection of all the maximal unqualified set.
- Setup: Run $\mathsf{Setup}(1^k)$ as normal signature, all the server and user get pp.
- $\mathsf{Keygen}(pp)$: A trusted dealer selects $g \xleftarrow{\$} \mathbb{G}_1^*, \tilde{g} \xleftarrow{\$} \mathbb{G}_2^*$ and $(x,y,K) \xleftarrow{\$} \mathbb{Z}_p^3$, computes $(X,Y) \leftarrow (g^x, g^y), (\tilde{X}, \tilde{Y}) \leftarrow (\tilde{g}^x, \tilde{g}^y)$. Set $\mathsf{sk} \leftarrow (x,X)$ and $\mathsf{pk} \leftarrow (g, Y, \tilde{g}, \tilde{X}, \tilde{Y}, \{\tilde{X}_T\}_{T \in \overline{\mathscr{T}}})$. Distribute x and K as (n,t)-threshold replicated share to $\{P_1, ..., P_n\}$ with $n \geq 3t+1$.
- Input to server P_j: Replicated share r_T^x for $\mathsf{sk} : x$, replicated share r_T^K for random value K.
- Input to U: Message $m \in \mathbb{Z}_p$.
  1. U choose $2t+1$ server (these server also willing to sign for the user) from $\{P_1, ..., P_n\}$, denote them as $\{S_1, ..., S_{2t+1}\}$.
  2. U commits on the message by computing $\mathsf{C} \leftarrow g^{\omega} Y^m$ where $\omega \xleftarrow{\$} \mathbb{Z}_p$. Send the commitment C to all the chosen server $\{S_1, ..., S_{2t+1}\}$ and prove that he knows message inside C if asked. Store ω for future unblinding.
  3. After receiving the commitment C , U checks if $C = 1_{\mathbb{G}_1}$ then abort. For any $(T, T') \in \overline{\mathscr{T}} \times \overline{\mathscr{T}}$ with $j \notin T$ and $j \notin T'$ server P_j computes $r_{T'}^u = \psi_{r_T^K}(\mathsf{C})$ and sends $(g^{r_{T'}^u}, g^{r_T^x \cdot r_{T'}^u}, \mathsf{C}^{r_{T'}^u})$ together with the index T, T' to U.
  4. After receiving$(g^{r_{T'}^u}, g^{r_T^x \cdot r_{T'}^u}, \mathsf{C}^{r_{T'}^u})$ and the index T, T' from P_j, U checks if $e(g^{r_{T'}^u}, \tilde{X}_T \cdot \tilde{Y}^m) = e(g^{r_T^x \cdot r_{T'}^u} \mathsf{C}^{r_{T'}^u} / g^{r_{T'}^u \cdot \omega}, \tilde{g})$. If fail, U knows that P_j is corrupted, and abort. (U may restart the protocol without choosing P_j.)
  5. U reconstruct $g^{x \cdot u} = \prod_{(T,T') \in \overline{\mathscr{T}} \times \overline{\mathscr{T}}} g^{r_T^x \cdot r_{T'}^u}$, $g^u = \prod_{T' \in \overline{\mathscr{T}}} g^{r_{T'}^u}$,$\mathsf{C}^u = \prod_{T' \in \overline{\mathscr{T}}} \mathsf{C}^{r_{T'}^u}$ and get the signature $\sigma' \leftarrow (g^u, g^{x \cdot u} \cdot \mathsf{C}^u)$.
  6. U unblind the signature by computing $\sigma'' \leftarrow (\sigma_1', \sigma_2' / \sigma_1'^{\omega})$. Where ω was stored in step 2.
  7. U randomize σ'' and get the final signature $\sigma = ({\sigma''}_1^{\gamma}, {\sigma''}_2^{\gamma})$ where $\gamma \xleftarrow{\$} \mathbb{Z}_p$.

The above protocol still has a room for optimization. Like the communication between U and server can be compressed into one round, and the share can be aggregated to save communication cost. It's not necessary to let each server send $g^{r_{T'}^u}$ and $\mathsf{C}^{r_{T'}^u}$ as there is repetitions.

In step 4, each chosen sever P_j locally compute a replicated share $r_{T'}^u$ for some random secret u through a PRF ψ. Note that the commitment C plays as a counter here, it will make sure different server sync the share of u for each signature.

The reason we need $n \geq 3t+1$ and choose $2t+1$ server is that to compute $g^{r_T^x \cdot r_{T'}^u}$, we

need for each pair (T, T') there exists chosen server P_j with $j \notin T \wedge j \notin T'$. Since one index T would cover t server, two index would cover $2t$ sever, thus, we need at least $2t+1$ server to finish the signature.

3.2.2 Security Analysis.

Unforgeability As the process of the threshold blind signature is equivalent with the signing oracle $\mathscr{O}$ in PS assumption 1, the *Unforgeability* of the entire signature can be reduced to the PS assumption 1. For the partial secret key the adversary can not get from corrupted server, the *Unforgeability* of corresponding partial signature can also be reduced to the PS assumption 1.

Robustness. For each partial signature, the user U checks

$$e(g^{r^u_{T'}}, \tilde{X}_T \cdot \tilde{Y}^m) = e(g^{r^x_T \cdot r^u_{T'}} \mathsf{C}^{r^u_{T'}} / g^{r^u_{T'} \cdot \omega}, \tilde{g})$$

. If the corresponding server use the right share of secret key x, this will be true, otherwise, it can not pass the check. When it fail, the user knows that this server is corrupted, and may restart the protocol without choosing this server. As long as there is $t+1$ honest server, U still can get a valid signature. Therefore, we have Robustness on our protocol above. Note that all the check in step 5 guarantee that the final signature is a valid signature, thus, it's not necessary for U to do a final check after step 9.

Static Semantic Security. As the commitment C perfectly hides the message m and the final signature is randomized before given to the adversary, no adversary can learn non-negligible information of m from C and σ. Therefore, for any adversary $\mathsf{Exp}^{\mathsf{SSS}}_{\mathscr{A}}(k, \mathsf{n}, \mathsf{t}) = 1] \approx_k 1/2$, which means we have Static Semantic Security on our protocol above.

Partial Signature Simulatability. The most powerful adversary can corrupt t servers which form a maximal unqualified subset of n servers. Thus, the adversary lacks only one partial signature $(g^{r^u_\tau}, g^{r^x_\tau \cdot r^u_\tau}, \mathsf{C}^{r^u_\tau})$ to construct the final signature. We use $\mathscr{T}'$ to denote the the set of index of partial signature the adversary could get from corrupted server and τ to denote the only index not in $\mathscr{T}'$.

SimPart[1]

1. Take input as $(n, t, C, \sigma, \omega, \{(g^{r^u_{T'}}, g^{r^x_T \cdot r^u_{T'}}, \mathsf{C}^{r^u_{T'}})\}_{(T,T') \in \mathscr{T}' \times \mathscr{T}'})$.
2. Reconstruct $\sigma' = (\sigma_1^\beta, \sigma_2^\beta \sigma_1^{\omega \cdot \beta})$ where $\beta \xleftarrow{\$} \mathbb{Z}_p$.
3. Compute $g^{r^u_\tau} \leftarrow \sigma'_1 / \prod_{T \in \mathscr{T}'} g^{r^u_T}$.
4. Compute $g^{r^x_\tau \cdot r^u_\tau} \xleftarrow{\$} \mathbb{G}_1 / 1_{\mathbb{G}_1}$.
5. Compute $\mathsf{C}^{r^u_\tau} \leftarrow \sigma'_2 / (g^{r^x_\tau \cdot r^u_\tau} \prod_{T' \in \mathscr{T}'} \mathsf{C}^{r^u_{T'}} \prod_{(T,T') \in \mathscr{T}' \times \mathscr{T}'} g^{r^x_T \cdot r^u_{T'}})$.
6. Return $(g^{r^u_\tau}, g^{r^x_\tau \cdot r^u_\tau}, \mathsf{C}^{r^u_\tau})$.

We argue that output of $\mathsf{SimPart}^1$ has same probability distribution with the partial signature output by honest server in real process of the scheme. By choosing random β, it is actually equivalent with choosing different u and different γ, as the adversary has zero information on u and γ, it's not possible to distinguish between output of $\mathsf{SimPart}^1$ and partial signature output by honest server. Thus, we get a PPT algorithm $\mathsf{SimPart}$ s.t. for any PPT adversary $\mathscr{A}$, $\Pr[\mathsf{Exp}^{\mathsf{PSS}}_{\mathscr{A}}(k,\mathsf{n},\mathsf{t})=1] \approx_k 1/2$. Therefore, the above scheme has Partial Signature Simulatability.

3.2.3 Efficiency Analysis

It's easy to see that we create $\binom{n}{t}$ share for each secret. To compute partial signature, $\binom{n}{t}\binom{n}{t}$ communication cost is required, which is really inefficient. Even if there is room for optimization, $\binom{n}{t}\binom{n}{t}$ overhead is necessary since we can not let U first learn g^u and then $(g^u)^x$ which let server learn g^u.

3.3 TBS based on Shamir's Secret Sharing

Compared with replicated secret share, Shamir's secret share is much more efficient as each player only have to hold a share same size with the secret. While the disadvantage is that Shamir's share can not play as input to PRF, since the output is not guaranteed to stay on one polynomial. Thus, we need to introduce one more round of communication between user and all the server to update the share for a new random $u \in \mathbb{Z}_q$.

3.3.1 The Protocol

With Shamir's secret share on secret key x and a random value u, each server can compute a partial signature, then it's easy for U to check if the partial signature is correct and construct the final signature with Lagrange interpolation.

Threshold PS signature based on Shamir's secret-share

- Invariant: $x = f(0)$, $u = f'(0)$, $\tilde{X}_i = \tilde{g}^{f(i)}$, $i \in [n]$, which is the set of all server. $l_i = \prod_{j \in S, j \neq i} \frac{j}{j-i}$, $l'_i = \prod_{j \in S', j \neq i} \frac{j}{j-i}$.
- Setup: Run $\mathsf{Setup}(1^k)$ as normal signature, all the server and user get pp.
- $\mathsf{Keygen}(pp)$: A trusted dealer selects $g \xleftarrow{\$} \mathbb{G}_1^*, \tilde{g} \xleftarrow{\$} \mathbb{G}_2^*$ and $(x, y, u) \xleftarrow{\$} \mathbb{Z}_p^3$, computes $(X, Y) \leftarrow (g^x, g^y), (\tilde{X}, \tilde{Y}) \leftarrow (\tilde{g}^x, \tilde{g}^y)$. Set $\mathsf{sk} \leftarrow (x, X)$ and $\mathsf{pk} \leftarrow (g, Y, \tilde{g}, \tilde{X}, \tilde{Y}, \{\tilde{X}_i\}_{i \in [n]})$. Distribute x, u as (n, t)-threshold Shamir's secret share to n server like $[x; f]_t = \{f(1), f(2), ..., f(n)\}, [u; f']_t = \{f'(1), f'(2), ..., f'(n)\}$ with $n \geq 3t + 1$.
- Input to server P_i: Shamir's share $f(i), f'(i)$ for secret x, u.
- Input to U: Message $m \in \mathbb{Z}_p$.
  1. U randomly choose a subset $S' \subset [n]$ with $|S'| = 2t + 1$ and denote the set of first $t + 1$ element in S' as S.
  2. U commits on the message by computing $\mathsf{C} \leftarrow g^{\omega} Y^m$ where $\omega \xleftarrow{\$} \mathbb{Z}_p$. Send the commitment C to all the chosen server $\{P_i\}_{i \in S'}$ and prove that he knows message inside C if asked.
  3. After receiving the commitment C, sever P_i sends $(g^{f'(i)}, g^{f(i)f'(i)}, \mathsf{C}^{f'(i)})$ to U.
  4. After receiving$(g^{f'(i)}, g^{f(i)f'(i)}, \mathsf{C}^{f'(i)})$ from P_i, U checks if $e(g^{f'(i)}, \tilde{X}_i \cdot \tilde{Y}^m) = e(g^{f(i)f'(i)} \mathsf{C}^{f'(i)} / g^{f'(i) \cdot \omega}, \tilde{g})$. If fail, U knows that P_i is corrupted, and abort. (U may restart the protocol without choosing P_i.)
  5. U reconstruct $g^{x \cdot u} = \prod_{i \in S'} g^{f(i)f'(i)l'_i}$, $g^u = \prod_{i \in S} g^{f'(i)l_i}$,$\mathsf{C}^u = \prod_{i \in S} \mathsf{C}^{f'(i)l_i}$ and get the signature $\sigma' \leftarrow (g^u, g^{x \cdot u} \cdot \mathsf{C}^u)$.
  6. For each $i \in [n]$, U prove to P_i that he knows $g^{f'(i)}$ with zero-knowledge proof. (U can compute $g^{f'(i)}$ with Lagrange inetpolation.)
  7. If server P_i is convinced that U knows $g^{f'(i)}$, P_i ask the trusted dealer to distribute Shamir's share for a new random value u'.
  8. U unblind the signature by computing $\sigma'' \leftarrow (\sigma'_1, \sigma'_2 / \sigma'^{\omega}_1)$.
  9. U randomize σ'' and get the final signature $\sigma = (\sigma''^{\gamma}_1, \sigma''^{\gamma}_2)$ where $\gamma \xleftarrow{\$} \mathbb{Z}_p$.

There is still room for optimization, not every chosen server has to send $(g^{f'(i)}, \mathsf{C}^{f'(i)})$.

Unforgeability As the process of the threshold blind signature is equivalent with the signing oracle $\mathcal{O}$ in PS assumption 1, the *Unforgeability* of the entire signature can be reduced to the PS assumption 1. For the partial secret key the adversary can not get from corrupted server, the *Unforgeability* of corresponding partial signature can also be

reduced to the PS assumption 1.

3.3.2 Security Analysis

Unforgeability As the process of the threshold blind signature is equivalent with the signing oracle $\mathscr{O}$ in PS assumption 1, the *Unforgeability* of the entire signature can be reduced to the PS assumption 1. For the partial secret key the adversary can not get from corrupted server, the *Unforgeability* of corresponding partial signature can also be reduced to the PS assumption 1.

Robustness For each partial signature, the user U checks

$$e(g^{f'(i)}, \tilde{X}_i \cdot \tilde{Y}^m) = e(g^{f(i)f'(i)} \mathsf{C}^{f'(i)} / g^{f'(i)\cdot\omega}, \tilde{g})$$

. If the corresponding server use the right share of secret key x, this will be true, otherwise, it can not pass the check. When it fail, the user knows that this server is corrupted, and may restart the protocol without choosing this server. As long as there is $2t+1$ honest server, U still can get a valid signature. Therefore, we have Robustness on our protocol above. Note that all the check in step 5 guarantee that the final signature is a valid signature, thus, it's not necessary for U to do a final check after step 9.

Static Semantic Security. As the commitment C perfectly hides the message m and the final signature is randomized before given to the adversary, no adversary can learn non-negligible information of m from C and σ. Therefore, for any adversary $\mathsf{Exp}^{\mathsf{SSS}}_{\mathscr{A}}(k, \mathsf{n}, \mathsf{t}) = 1] \approx_k 1/2$, which means we have Static Semantic Security on our protocol above.

Partial Signature Simulatability The most powerful adversary can corrupt t servers which form a maximal unqualified subset of n servers. Thus, the adversary lacks only one partial signature $(g^{f'(\tau)l_\tau}, g^{f(\tau)f'(\tau)l'_\tau}, \mathsf{C}^{f'(\tau)l_\tau})$ to construct the final signature. We use C to denotes the the set of index of partial signature the adversary could get from corrupted server and τ to denote the only index not in C.

SimPart^2

1. Take input as $(n, t, C, \sigma, t', \{(g^{f'(i)l_i}, g^{f(i)f'(i)l'_i}, \mathsf{C}^{f'(i)l_i})\}_{i\in C})$.
2. Reconstruct $\sigma' = (\sigma_1^\beta, \sigma_2^\beta \sigma_1^{\omega\cdot\beta})$ where $\beta \xleftarrow{\$} \mathbb{Z}_p$.
3. Compute $g^{f'(\tau)l_\tau} \leftarrow \sigma'_1 / \prod_{i\in C} g^{f'(i)l_i}$.
4. Compute $g^{f(\tau)f'(\tau)l'_\tau} \xleftarrow{\$} \mathbb{G}_1 / 1_{\mathbb{G}_1}$.
5. Compute $\mathsf{C}^{f'(\tau)l_\tau} \leftarrow \sigma'_2 / g^{f(\tau)f'(\tau)l'_\tau} \prod_{i\in C} \mathsf{C}^{f'(i)l_i} \prod_{i\in C} g^{f(i)f'(i)l'_i}$.
6. Return $(g^{f'(\tau)l_\tau}, g^{f(\tau)f'(\tau)l'_\tau}, \mathsf{C}^{f'(\tau)l_\tau})$.

We argue that output of SimPart^2 has same probability distribution with the partial signature output by honest server in real process of the scheme. By choosing random β,

it is actually equivalent with choosing different u and different γ, as the adversary has zero information on u and γ, it's not possible to distinguish between output of $\mathsf{SimPart}^2$ and partial signature output by honest server. Thus, we get a PPT algorithm $\mathsf{SimPart}$ s.t. for any PPT adversary $\mathscr{A}$, $\Pr[\mathsf{Exp}^{\mathsf{PSS}}_{\mathscr{A}}(k,\mathsf{n},\mathsf{t})=1] \approx_k 1/2$. Therefore, the above scheme has Partial Signature Simulatability.

Above security analysis focus on one round of signature. About updating the share of random u, the reason is same. Even if an adversary with share from t corrupted server, he can not reconstruct the polynomial f' and convince all the server, thus, it must be the case that a new signature just finished.

3.3.3 Efficiency Analysis

Clearly, using Shamir's secret share makes our protocol much more efficient and near optimal both in time and space complexity. The share for each server has same size with the secret, to compute partial signature, each server also only needs constant operation on the group.
But the disadvantage compares to replicated share is also clear, as we don't want allow communication between server, it's not possible to locally generate new Shamir's share for new random secret from Shamir's share for old value. Thus, it always needs a trusted dealer to distribute Shamir's share for new random u, which brings more risks. Remember that, based on replicated secret share, only at the very beginning we need a trusted dealer.

3.4 TBS on Multiple Message with Share Conversion

In this section, we present our threshold blind signature scheme based on the multi-message protocol in 2.5.2. From above two sections, we are clear about the advantage and disadvantage of using replicated secret share or Shamir's secret share for the secret key and random value of our threshold PS signature scheme.
When using replicated secret share, we need $O(\binom{n}{t})$ space for storing the share, and $O(\binom{n}{t}\binom{n}{t})$ communication cost for partial signature, which is really inefficient, but trusted dealer is only needed at the very beginning when configuring all the server.
When using Shamir's secret share, we are near optimal both for storing the share and computing or sending partial signature, but we always need a trusted dealer to distribute share for random value u since Shamir's share can not play as key input for a PRF ψ and we don't want to allow communication between server.
It's natural to ask, is it possible to enjoy best in both worlds? The answer is yes, with local share conversion proposed in [CDI05], we can enjoy the advantage of replicated share and Shamir's share at same time.

3.4.1 The Protocol

With PRSS, if all the server agreed on a value, and they have replicated share on some secret, they can locally generate replicated share for infinite new random secret which is what it cost for infinite threshold PS signature. But storing replicated share for a lot of random secret is space inefficient and even impossible. The idea is to store only one

replicated share for K as key input to PRF ψ and use share conversion to convert other replicated share into Shamir's share locally.
In other words, with share conversion, the server can prepare Shamir's share for new random $u \in \mathbb{Z}_p$ in a off-line mode, no communication with other server nor trusted dealer is needed after each server is configured at the very beginning.

Threshold PS signature based on Share Conversion

- Invariant: $x = f(0)$, $u = f'(0)$, $\tilde{X}_i = \tilde{g}^{f(i)}$, $i \in [n]$, which is the set of all server. $l_i = \prod_{j \in S, j \neq i} \frac{j}{j-i}$, $l'_i = \prod_{j \in S', j \neq i} \frac{j}{j-i}$.
- Setup: Run $\mathsf{Setup}(1^k)$ as normal signature, all the server and user get pp.
- $\mathsf{Keygen}(pp)$: A trusted dealer selects $(g, \tilde{g}) \xleftarrow{\$} (\mathbb{G}_1^*, \mathbb{G}_2^*)$ and $(K, x, y_1, y_2, ..., y_r) \xleftarrow{\$} \mathbb{Z}_p^{r+2}$, computes $(X, Y_1, Y_2, ..., Y_r) \leftarrow (g^x, g^{y_1}, g^{y_2}, ..., g^{y_r}), (\tilde{X}, \tilde{Y}_1, \tilde{Y}_2, ..., \tilde{Y}_r) \leftarrow (\tilde{g}^x, \tilde{g}^{y_1}, \tilde{g}^{y_2}, ..., \tilde{g}^{y_r})$. Set $\mathsf{sk} \leftarrow (x, X)$ and $\mathsf{pk} \leftarrow (g, \{Y_i\}_{i \in [r]}, \tilde{g}, \tilde{X}, \{\tilde{Y}_i\}_{i \in [r]}, \{\tilde{X}_i\}_{i \in [n]})$. Distribute x as (n,t)-threshold Shamir's secret share to n server like $[x; f]_t = \{f(1), f(2), ..., f(n)\}$ and K as (n,t)-threshold replicated share to $\{P_1, ..., P_n\}$ with $n \geq 3t+1$.
- Input to server P_i: Shamir's share $f(i)$ for secret key x, replicated share r_T^K for random value K.
- Input to U: Multiple message $m_i \in \mathbb{Z}_p$ for $i \in [r]$.
  1. U randomly choose a subset $S' \subset [n]$ with $|S'| = 2t+1$ and denote the set of first $t+1$ element in S' as S.
  2. U commits on the messages by computing $\mathsf{C} \leftarrow g^{\omega} \prod_{i \in [r]} Y_i^{m_i}$. Send the commitment C to all the chosen server $\{P_i\}_{i \in S'}$ and prove that he knows message inside C if asked.
  3. After receiving the commitment C , sever P_i computes $r_T^u = \psi_{r_T^K}(\mathsf{C})$ for $j \notin T$ and use share conversion get $f'(i)$ locally, sends $(g^{f'(i)}, g^{f(i)f'(i)}, \mathsf{C}^{f'(i)})$ to U.
  4. After receiving$(g^{f'(i)}, g^{f(i)f'(i)}, \mathsf{C}^{f'(i)})$ from P_i, U checks if $e(g^{f'(i)}, \tilde{X}_i \cdot \prod_{i \in [r]} \tilde{Y}_i^{m_i}) = e(g^{f(i)f'(i)} \mathsf{C}^{f'(i)} / g^{f'(i) \cdot \omega}, \tilde{g})$. If fail, U knows that P_i is corrupted, and abort. (U may restart the protocol without choosing P_i.)
  5. U reconstruct $g^{x \cdot u} = \prod_{i \in S'} g^{f(i)f'(i)l'_i}$, $g^u = \prod_{i \in S} g^{f'(i)l_i}$,$\mathsf{C}^u = \prod_{i \in S} \mathsf{C}^{f'(i)l_i}$ and get the signature $\sigma' \leftarrow (g^u, g^{x \cdot u} \cdot \mathsf{C}^u)$.
  6. U unblind the signature by computing $\sigma'' \leftarrow (\sigma'_1, \sigma'_2 / \sigma'^{\omega}_1)$.
  7. U randomize σ'' and get the final signature $\sigma = (\sigma''^{\gamma}_1, \sigma''^{\gamma}_2)$ where $\gamma \xleftarrow{\$} \mathbb{Z}_p$.

In the above protocol, the server prepare Shamir's share for u when it is asked to compute partial signature, to make it more efficient, server can prepare Shamir's share

for u in a off-line mode, but it will cost more communication resources since all the server needs to sync on the share of u.

3.4.2 Security Analysis

Unforgeability The protocol above is equivalent to generate a valid signature same as protocol in 2.5.2, thus, the *Unforgeability* can also be reduced to PS assumption 1, as proved in the appendix of [PS16].

Robustness For each partial signature, the user U checks

$$e(g^{f'(i)}, \tilde{X}_i \cdot \prod_{i \in [r]} \tilde{Y}_i^{m_i}) = e(g^{f(i)f'(i)} \mathsf{C}^{f'(i)} / g^{f'(i) \cdot \omega}, \tilde{g})$$

. If the corresponding server use the right share of secret key x, this will be true, otherwise, it can not pass the check. When it fail, the user knows that this server is corrupted, and may restart the protocol without choosing this server. As long as there is $2t+1$ honest server, U still can get a valid signature. Therefore, we have Robustness on our protocol above. Note that all the check in step 4 guarantee that the final signature is a valid signature, thus, it's not necessary for U to do a final check after step 9.

Static Semantic Security. As the commitment C perfectly hides the message m and the final signature is randomized before given to the adversary, no adversary can learn non-negligible information of m from C and σ. Therefore, for any adversary $\mathsf{Exp}^{\mathsf{SSS}}_{\mathscr{A}}(k, \mathsf{n}, \mathsf{t}) = 1] \approx_k 1/2$, which means we have Static Semantic Security on our protocol above.

Partial Signature Simulatability As the partial signature has same struct as partial signature in 3.3.1, it can also be simulated by $\mathsf{SimPart}^2$, thus, we have *Partial Signature Simulatability* on the protocol above.

3.4.3 Efficiency Analysis

As proved in [CDI05], $\binom{n}{t}$ overhead is inevitable when converting replicated share into Shamir's share locally. Thus, it will significantly slow down the speed of computing partial signature.
Another difference compared with Threshold PS signature based on Shamir's secret-share is that each server always need to store replicated share for at least one random secret. Which takes $O(\binom{n-1}{t})$ space for each server.
But as we don't need communication to make all the server sync on the share of u, the communication cost is near optimal, which makes this protocol to be the best choice in general.

3.5 Proving Knowledge of a Signature Multiple Message

The above sections solve the problem of issuing threshold PS signature without revealing the signing message m. In some scenario, user may wants to prove that he has

some valid signature without revealing any information of the signed message. In more special case, the user may wants to prove partial information of the signed message. As the final signature σ was randomized by the user, it would be safe to directly send σ to the verifier, and prove with zero-knowledge that he has the set of message and randomness satisfying the verification pairing formula.

Chapter 4

Implementation

In this chapter, we present the implementation of our threshold blind signature scheme based on BLS12-381 curve and evaluate performance from different aspects. In general, our goal is to test the performance not to do some attack and test for security, we will take every participant as honest.
Bilinear form looks very abstract at first glance, it's natural to ask how it is implemented in real world. Fortunately, there is some public resource for BLS (Barreto, Lynn and Scott) curve [BLS03] and BLS(Boneh, Lynn and Shacham) signature [BLS01] namely the open source project *blst*[1]. As we mentioned in 7, certain subgroup of BLS curve satisfies the SXDH assumption and thus can be used to implement PS signature scheme and threshold PS signature scheme.
In the following sections, we will first present the implementation of operation on BLS12-381, and implementation of protocols in Chapter 3, finally, we will test performance from different aspects.
Since Go has good supports for concurrency and produce relatively efficient executable, we choose to use Go for this implementation.

4.1 Implementation of Bilinear Group

Curve BLS12-381 was designed by Sean Bowe in early 2017 as the foundation for an upgrade to the Zcash protocol. BLS12-381 is designed to achieve 128bits security level and widely used in digital signature and other pairing based cryptosystem. As it is the most efficient and mature curve which has subgroups satisfying SXDH assumption, we can use it to build the bilinear group for our threshold blind signature scheme.

blst provides a mature and efficient implementation of curve BLS12-381 and BLS signature scheme. It is written in C and assembly but also provides API for other language like Go and Rust. Since their major goal is to implement BLS signature scheme, their API for Go not include basic operation on BLS curve and pairing. Thus, we need to add some API for basic operation on the BLS curve. We create a file *BLS_curve.go*[2] for our API. The API we created is as below

[1] An implementation of BLS signature can be found at https://github.com/supranational/blst
[2] under blst/bindings/go/

APIs in *BLS_curve.go*

- Basic operation on $\mathbb{Z}_p$ including
  1. generate a random element in $\mathbb{Z}_p$.
  2. add, multiply two element in $\mathbb{Z}_p$.
  3. compute inverse of element in $\mathbb{Z}_p$.
  4. Convert short integer into element in $\mathbb{Z}_p$.
- Basic operation on $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ including
  1. Return the generator $g, \tilde{g}$.
  2. Multiply two element.
  3. Compute h^x where $x \in \mathbb{Z}_p$.
  4. Hash an element in $\mathbb{G}_1$ or $\mathbb{G}_2$ to element in $\mathbb{Z}_p$.
- Compute the map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.

With APIs above, we can easily compute any algebraic operation of bilinear group, thus, we can directly implement a simple version of PS signature scheme for single message.

4.2 Implementation of Our Proposed Schemes

To implement our threshold blind signature scheme, more things need to be considered like how to simulate the trusted dealer, server and user, how to simulate communication between them.
Our first idea is to deploy our signer and dealer on some cloud server and let them communicate with socket by encoding every element as bytes. This is a good model as it is closest to real application scenarios. But it will introduce the noise of internet, since the communication cost is determined and predictable, It's not interest to measure the time for communication since it majorly depends on the bandwidth.
As our goal is to test for efficiency, it would be easier to simulate server on local machine, we can use *Goroutine* to simulate one server, but it will make no difference since the underlying implementation of curve operation will exploit all the physical threads. Our final solution is as below.

Trusted Dealer In our protocol, the trusted dealer would generate secret and public key and distribute them as secret share to all the server. We implement this in the function *Keygen*() called by main function. It will write share to each server and user.

Server and User After the *Keygen*() finished, main function will write a random message $m \in \mathbb{Z}_p$ to U, and let U start commit on the message and call the method of *server* for signature.

With these idea in mind, the last thing we need is to implement replicated secret share, Shamir's secret share, Lagrange interpolation, Share conversion. With basic

algebraic operation ready, most of them are easy to implement, we will present some important detail below.

Maximal Unqualified Set Remember in section 3.2.1, we use T to denote maximal unqualified set, this looks abstract at first glance, but easy to implement since we can use one bit to denote existence of one server, as it growth factorially, integer with 8 bytes is enough for this.
We create 4 files in main work place as *test.go* for main function and testing, *Replicated.go* for protocol in 3.2.1, *Shamir's.go* for protocol in 3.3.1, *Share_conversion.go* for protocol in 3.4.1.

4.3 Test and Performance

With protocols in Chapter 3 implemented, we can finally test for their performance.
We will test from two aspect, with different number of signer and different number of signature after one configuration. For each test, we will measure time usage in two phase. Namely *configuration* phase and *signing* phase.

Configuration In this phase, trusted dealer select the keys and distribute them as secret share to all the server. Each server will also do some local computation. After this phase is finished, each server will be ready for call from user.

Signing Before testing start, we already set the number of message waiting for signature. This phase including signing all the message and verification. Our code can be found on github[3].

4.3.1 Environment and Parameters

We run the code on Go version of *go1.16.3 darwin/amd64* with processor i7-9750H @2.6GHz and 32GB DDR4 RAM @2400MHz.
The security parameter is 1^{256} means we need 32 bytes for the secret key. For simplicity, we always set $n = 3t + 1$ and user always choose first $2t + 1$ servers.
As we test on local machine, but in real world, each server always has independent computing resources. When measuring time consuming, we divide it by the number of servers to simulate deployment in real world.

4.3.2 Test with Growing number of Server

In this section, we test for time consuming with growing number of servers. In each test, we do 5 signature.

[3] https://github.com/Heisenberge/Threshold-PS-signature-scheme

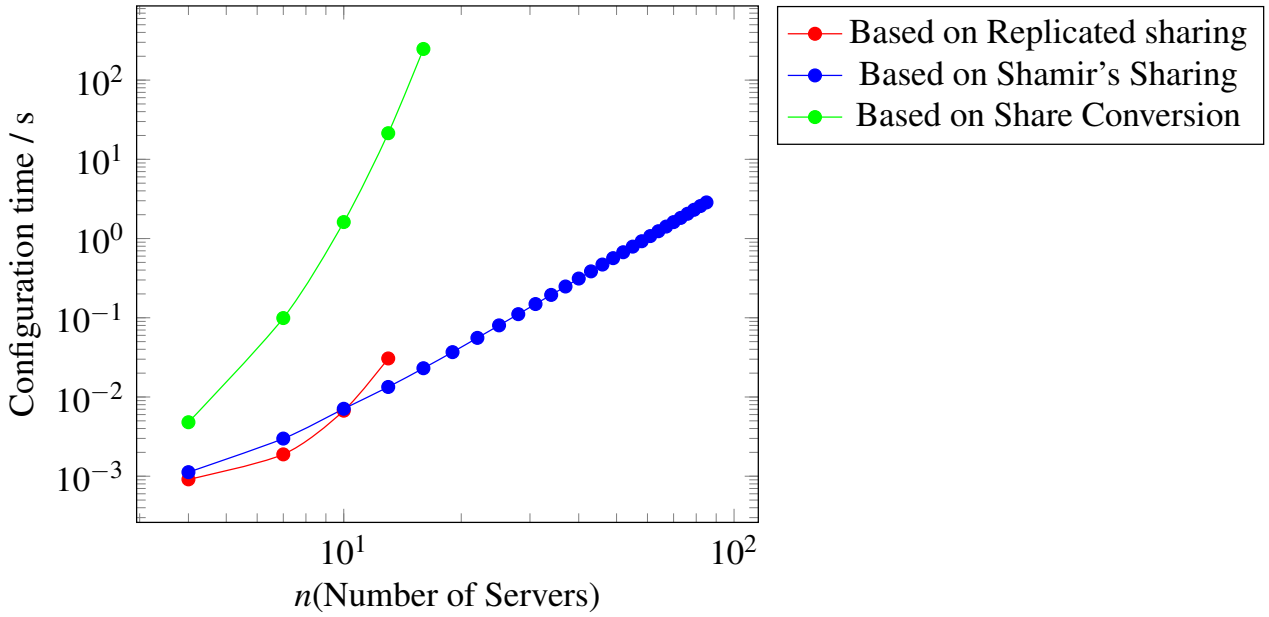

Figure 4.1: Configuration time with growing number of server

Configuration Time As we can see, our protocol based on share conversion takes the longest time in the configuration phase. This is because $O(\binom{n}{t})$ overhead is necessary in the share conversion and Lagrange interpolation is much slower than choosing random element in replicated secret sharing.

Signature Time As we can see, our protocol based on share conversion is most efficient in the signature phase. This is because it already prepared all the share of u. The protocol based on replicated secret sharing only works on very limited number of servers. If we allow server prepare in off-line mode, the protocol based on share conversion would be our best choice.

4.3.3 Test with Growing number of Signature

In this section, we test for time consuming with growing number of signature. We set $t = 2, n = 7$.

Configuration Time As we can see, the configuration time of protocol based on replicated sharing and Shamir's sharing is stable and independent with the number of signature while our protocol based on Share Conversion grows very fast with number of signature. This is because we prepare all the share of u at configuration phase, in real world, the server could do this in spare time or when the share for u is nearly exhausted.

Signature Time For the signature time, we can see our protocol based on share conversion is the fastest one and a little bit faster than the protocol based on Shamir's

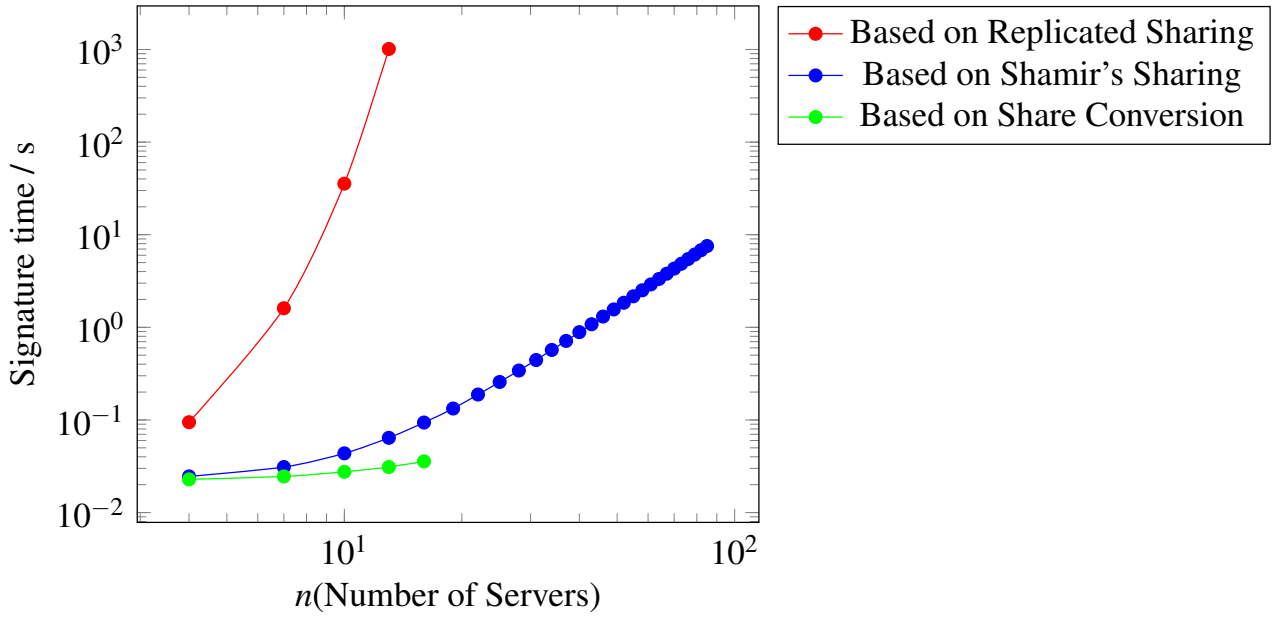

Figure 4.2: Signature time with growing number of server

share. This is because it already prepared all the share of u. The signature efficiency of our protocol based on replicated secret sharing is the worst.

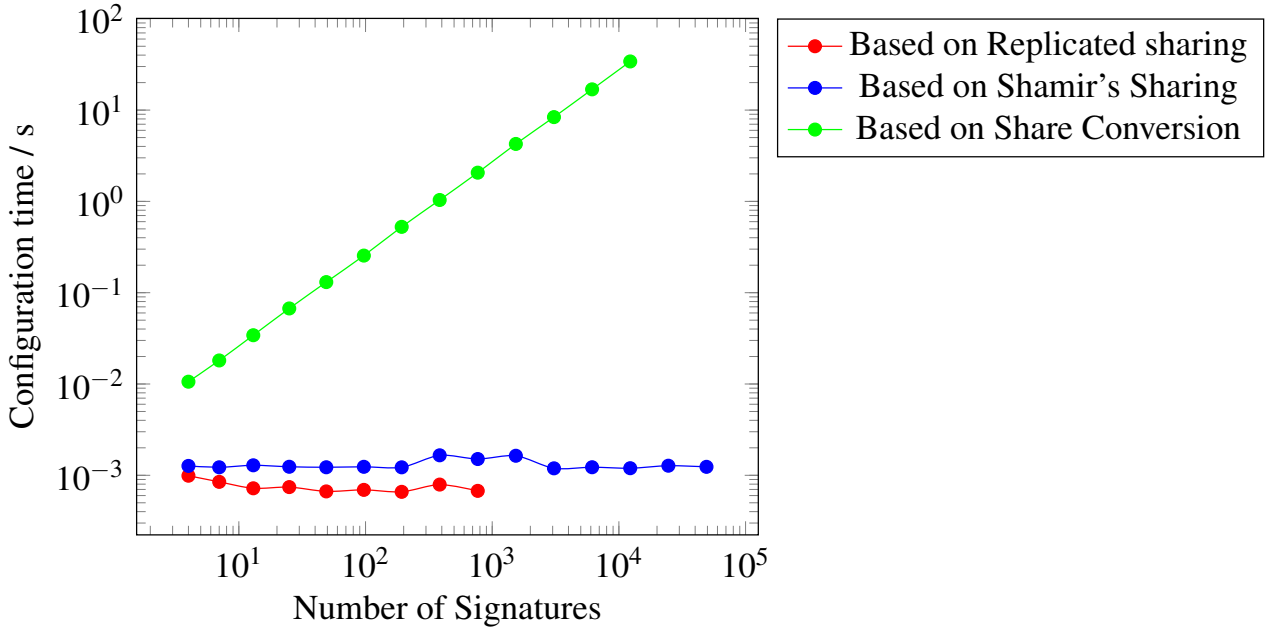

Figure 4.3: Configuration time with growing number of signature

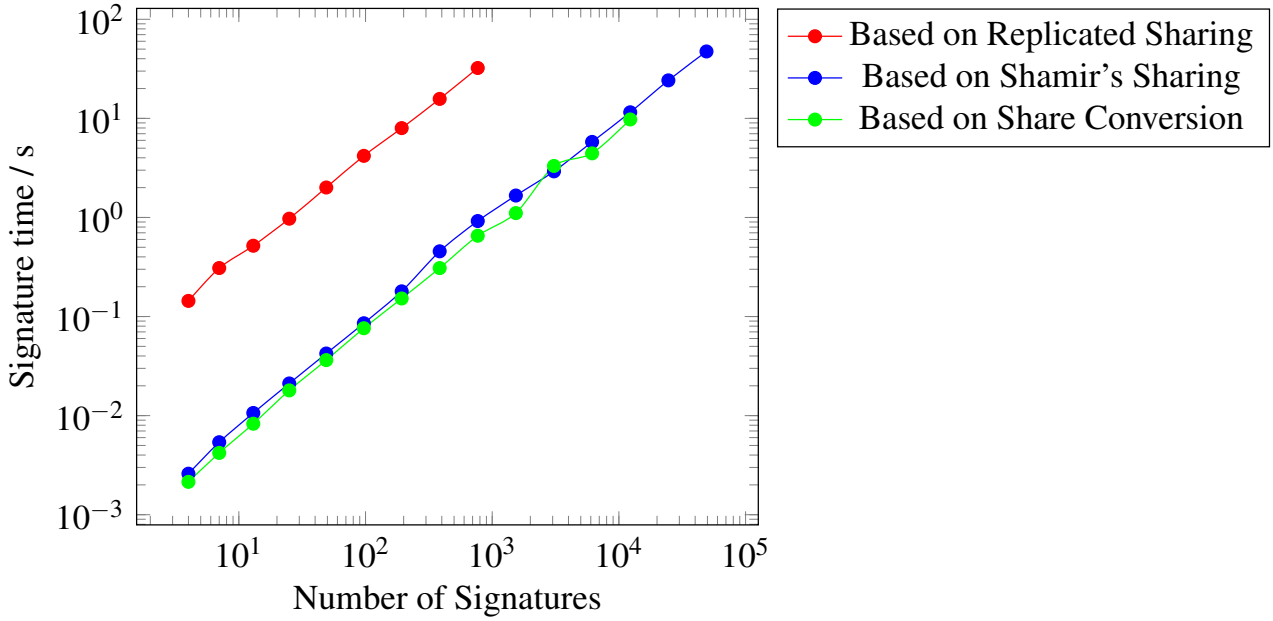

Figure 4.4: Signature time with growing number of signature

Chapter 5

Application

In this chapter, we present some application scenario of threshold blind signature scheme. The application of TBS comes from its two good property, on the one hand, it perfectly hides the message of signer, on the other hand, it hides the identity of signer actually produce the signature to some extent.

5.1 Anonymous Credential

Anonymous Credential System was first introduced by Chaum in [Cha83]. It is a general problem to protect privacy when getting credential from some authority agency. Like when issuing passport, the client may not want to expose all his personal information, but as it is a passport, the client may later needs to prove that his age is beyond 18. A general solution is to first commit on the message, let the authority sign on the commitment. Later, the client may use zero-knowledge proof to prove partial information of the signed message. Let's see a few specific application scenarios below.

e-Voting When a citizen casts a vote, he needs to prove his citizenship to the authority, but he wants to hide his vote. With blind signature, he can first commit on his vote, send the commitment with his authentication of citizenship to the authority. The authority check the authentication and sign on the commitment, it's not possible to learn any information on the vote. After receiving the signed commitment back, the citizen can unblind it and get a signed vote before casting it to the vote center. With threshold blind signature, we can let multiple authority to check the citizenship, each generate a partial signature on commitment, only if the citizen pass all the authority check, he can get a valid vote.

Accountability in Blockchain Due to anonymity, blockchain-based cryptocurrencies are sought after by capital. But for the same reason, it is difficult for cryptocurrency to become the legal tender of major economies. It is an interesting topic to achieve some accountability while keep anonymity of blockchain. A possible solution was given in [DGK+20], they use threshold encryption as while as blind signature scheme to construct a identity manage system on blockchain. with agree of enough number of authorities, any user's identity and usage record can be deanonymized.

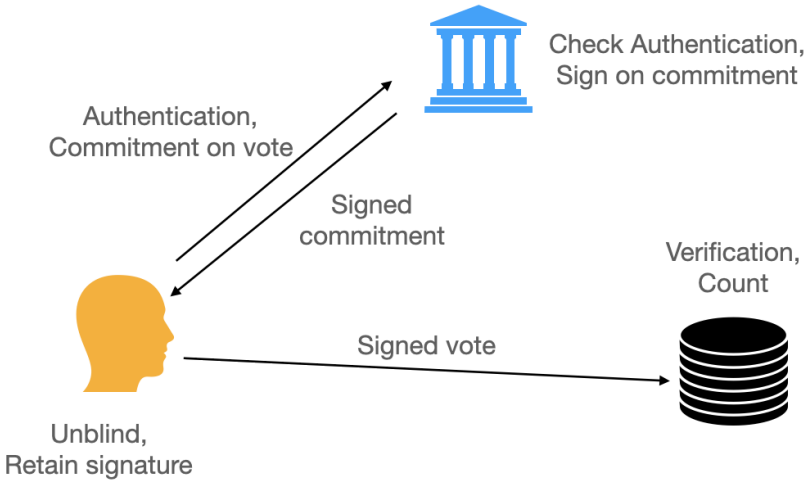

Figure 5.1: e-Voting with blind signature

5.2 Hiding the Actual Signer

In the (n,t)-threshold signature scheme, with secret sharing scheme, the sign ability is distributed to n server, with any $t+1$ server can generate a valid signature. Only the user and trusted dealer knows the actual signers for a signature.
Thus, the identity of the actual signer is hidden to a certain extent. Similar and even better property can be found in the ring signature scheme [RST01] in which the actual signer is perfectly hidden and group signature scheme [CvH91, CDL$^+$20] in which there exists a manager of identity of signers. Let's see some specific application scenario below.

Anonymous Voting In the e-Voting system, we need to hide the vote, in more special circumstances, we want to hide the identity of voter. With threshold signature scheme, we can distribute the share of secret key to all the possible voter, if a valid signature is generated, we know that at least $t+1$ votes on agree.

Leak a Secret Suppose a committee use a threshold signature scheme, each member has a share of the secret key. Some member of the committee wants to leak some private information to the public media. They have to prove the validity of the information but they don't want to expose their identity. If they form a set of $t+1$ members, they can generate a valid signature on the leaked information, which is as same as the signature from the committee. In such case, the actual signer is hidden to some extent.

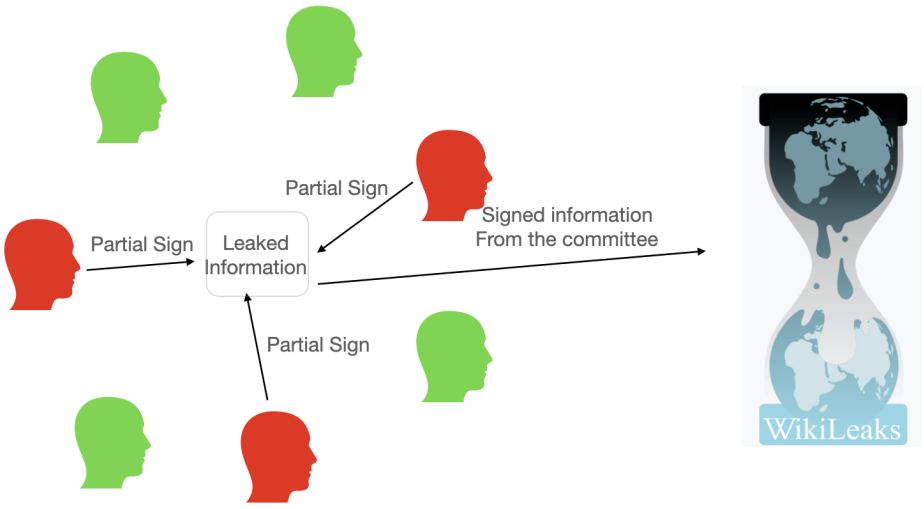

Figure 5.2: Leak a Secret

Chapter 6

Conclusion

In this thesis, we give our definition of threshold blind signature scheme, we construct threshold blind signature scheme based on the Pointcheval-Sanders signature scheme and analyse security and efficiency. Our construction used different secret sharing scheme and share conversion in [CDI05].
Based on *blst*, which implements BLS12-381 curve, we implement our 3 version of threshold blind PS signature scheme and test for their performance. We can conclude that, if we allow the server prepare shares in off-line mode, our threshold blind signature scheme based on share conversion would give us best performance in general.
Future work could be implementing protocols in chapter 3 on multiple messages on distributed systems to better simulate real world signature system. It is also possible to find more efficient curve than BLS12-381 which could form better bilinear group for our proposed scheme. Of course, there is some room for optimization in our proposed schemes.

Bibliography

[ADEO20] Diego ARANHA, Anders DALSKOV, Daniel ESCUDERO et Claudio ORLANDI : Improved threshold signatures, proactive secret sharing and input certification from lss homomorphisms. Cryptology ePrint Archive, Report 2020/691, 2020. https://eprint.iacr.org/2020/691.

[BLS01] Dan BONEH, Ben LYNN et Hovav SHACHAM : Short signatures from the weil pairing. *In* Colin BOYD, éditeur : *Advances in Cryptology — ASIACRYPT 2001*, pages 514–532, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[BLS03] Paulo S. L. M. BARRETO, Ben LYNN et Michael SCOTT : Constructing elliptic curves with prescribed embedding degrees. *In* Stelvio CIMATO, Giuseppe PERSIANO et Clemente GALDI, éditeurs : *Security in Communication Networks*, pages 257–267, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[BN06] Paulo S. L. M. BARRETO et Michael NAEHRIG : Pairing-friendly elliptic curves of prime order. *In* Bart PRENEEL et Stafford TAVARES, éditeurs : *Selected Areas in Cryptography*, pages 319–331, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[BOGG+90] Michael BEN-OR, Oded GOLDREICH, Shafi GOLDWASSER, Johan HÅSTAD, Joe KILIAN, Silvio MICALI et Phillip ROGAWAY : Everything provable is provable in zero-knowledge. *In* Shafi GOLDWASSER, éditeur : *Advances in Cryptology — CRYPTO' 88*, pages 37–56, New York, NY, 1990. Springer New York.

[CDI05] Ronald CRAMER, Ivan DAMGÅRD et Yuval ISHAI : Share conversion, pseudorandom secret-sharing and applications to secure computation. volume 3378, pages 342–362, 02 2005.

[CDL+20] Jan CAMENISCH, Manu DRIJVERS, Anja LEHMANN, Gregory NEVEN et Patrick TOWA : Short threshold dynamic group signatures. Cryptology ePrint Archive, Report 2020/016, 2020. https://eprint.iacr.org/2020/016.

[Cha83] David CHAUM : Blind signatures for untraceable payments. *In* David CHAUM, Ronald L. RIVEST et Alan T. SHERMAN, éditeurs : *Advances in Cryptology*, pages 199–203, Boston, MA, 1983. Springer US.

[CL04] Jan CAMENISCH et Anna LYSYANSKAYA : Signature schemes and anonymous credentials from bilinear maps. *In* Matt FRANKLIN, éditeur : *Advances in Cryptology – CRYPTO 2004*, pages 56–72, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[CvH91] David CHAUM et Eugène van HEYST : Group signatures. *In* Donald W. DAVIES, éditeur : *Advances in Cryptology — EUROCRYPT '91*, pages 257–265, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[DGK+20] Ivan DAMGÅRD, Chaya GANESH, Hamidreza KHOSHAKHLAGH, Claudio ORLANDI et Luisa SINISCALCHI : Balancing privacy and accountability in blockchain identity management. Cryptology ePrint Archive, Report 2020/1511, 2020. https://eprint.iacr.org/2020/1511.

[DH76] W. DIFFIE et M. HELLMAN : New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[GMW86] Oded GOLDREICH, Silvio MICALI et Avi WIGDERSON : Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. *In 27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 174–187, 1986.

[GPS08] Steven D. GALBRAITH, Kenneth G. PATERSON et Nigel P. SMART : Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16): 3113–3121, 2008. Applications of Algebra to Cryptography.

[GR04] Steven D. GALBRAITH et Victor ROTGER : Easy decision diffie-hellman groups. *LMS Journal of Computation and Mathematics*, 7:201Ð218, 2004.

[KM15] Veronika KUCHTA et Mark MANULIS : Rerandomizable threshold blind signatures. *In* Moti YUNG, Liehuang ZHU et Yanjiang YANG, éditeurs : *Trusted Systems*, pages 70–89, Cham, 2015. Springer International Publishing.

[LRSW00] Anna LYSYANSKAYA, Ronald L. RIVEST, Amit SAHAI et Stefan WOLF : Pseudonym systems. *In* Howard HEYS et Carlisle ADAMS, éditeurs : *Selected Areas in Cryptography*, pages 184–199, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[Nao90] Moni NAOR : Bit commitment using pseudo-randomness. *In* Gilles BRASSARD, éditeur : *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 128–136, New York, NY, 1990. Springer New York.

[PS16] David POINTCHEVAL et Olivier SANDERS : Short randomizable signatures. *In* Kazue SAKO, éditeur : *Topics in Cryptology - CT-RSA 2016*, pages 111–126, Cham, 2016. Springer International Publishing.

[PS17] David POINTCHEVAL et Olivier SANDERS : Reassessing security of randomizable signatures. Cryptology ePrint Archive, Report 2017/1197, 2017. https://eprint.iacr.org/2017/1197.

[RST01] Ronald L. RIVEST, Adi SHAMIR et Yael TAUMAN : How to leak a secret. *In* Colin BOYD, éditeur : *Advances in Cryptology — ASIACRYPT 2001*, pages 552–565, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[Sha79] Adi SHAMIR : How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[VZK03] Liem VO, Fangguo ZHANG et Kwangjo KIM : A new threshold blind signature scheme from pairings. 03 2003.