Distributed Multi-Party Fair Exchange

by

Aybüke Buket Akgül

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computer Science and Engineering

KOÇ ÜNİVERSİTESİ

April 24, 2024

Distributed Multi-Party Fair Exchange

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Aybüke Buket Akgül

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Assoc. Prof. Alptekin Küpçü (Advisor)

_____

Asst. Prof. Mehmet Emre Gürsoy

_____

Asst. Prof. Mehmet Tahir Sandıkkaya

Date: _____

To my family; my mother, my father and my cat...

ABSTRACT

Distributed Multi-Party Fair Exchange
Aybüke Buket Akgül
Master of Science in Computer Science and Engineering
April 24, 2024

Over the years, researchers focused on either two-party cases or multi-party computation securely, fairly and optimistically. Most of the studies in the literature require a trusted third party (TTP) to be present for fairness either at every step of the protocol or only optimistically intervene in case of malicious actions.

However, having a single TTP creates a single point of failure (in terms of security) in the protocol. To distribute this responsibility, literature has different solutions. Blockchain is one of the most used solutions for decentralizing exchange protocols and multi-party computations. Secret sharing is another procedure used for distributing responsibility, however, it is mostly used for 2-party settings.

As far as we know, there is no optimistic multi-party fair exchange solution using multiple TTPs in the literature. In this work, we present a Distributed Multi-Party Fair Exchange Protocol, building on top of a previous research [Alper and Küpçü, 2021], by distributing the responsibility of a single TTP to multiple (m) TTPs. We are using secret sharing to distribute the decryption share used for fairness to TTPs. Even with a malicious subset of TTPs, the protocol provides a fair result, as long as there are threshold-many honest TTPs. Even though the performance of the protocol is worse (slower, as expected) than the previous study, it carries the importance of being the first optimistic multi-party fair exchange protocol with multiple trusted third parties in the literature.

ÖZETÇE

Dağıtık Çok Kişilik Adil Takas
Aybüke Buket Akgül
Bilgisayar Bilimleri ve Mühendisliği, Yüksek Lisans
24 Nisan 2024

Adil Takas protokolleri, 2 veya daha fazla tarafın takas yapılan öğenin ya tüm taraflar tarafından elde edilebilir olduğu ya da hiçbirinin elde edemeği protokollerdir. Bu protokollerde adalet genellikle güvenilir bir üçüncü şahıs tarafından sağlanır. İlk çalışmalar güvenilir üçüncü şahsın protokolün bütün adımlarında dahil olduğu protokollerdir. Sonrasında "iyimser" dediğimiz, güvenilir üçüncü şahsın sadece bir problem çıktığı durumda dahil olduğu protokoller sunulmuştur.

Güvenilir üçüncü şahsın bulunması, güvenlik açısından protokolde tek arıza noktası yaratmaktadır. Bu sorumluluğun dağılması için literatürde farklı çalışmalar bulunmaktadır. Blok zincir uygulaması, özellikle çok taraflı adil takas protokollerinde en yaygın kullanılan yöntemlerden biridir. Ayrıca sır paylaşımı gibi methodlar da kullanılmaktadar; ancak bu çalışmalar genellikle iki taraflı adil takas protokollerinde görülmektedir.

Çok taraflı adil takas protokollerinde tek güvenilir üçüncü şahıs yerine birden fazla güvenilir "iyimser" üçüncü şahıs kullanılan bir çalışma şu anda literatürde bulunmamaktadır. Biz bu çalışmada tek güvenilir üçüncü şahıs yerine birden fazla (m tane) şahsın bulunduğu bir çok kişilik adil takas protokolü sunuyoruz. Protokolümüzün temeli, daha önce tek güvenilir üçüncü şahısla çalışan Alper ve Küpçü tarafından öne sunulan çok taraflı güvenilir ve adil takas protokolüne [Alper and Küpçü, 2021] dayanmaktadır. Biz bu protokoldeki, güvenilir üçüncü şahsın adaleti sağlaması için paylaşılan şifrelenmiş kilit açma anahtarını birden fazla (m tane) şahısa sır paylaşımı methodunu kullanarak paylaştırıyoruz. Bu şekilde şahıslardan bazıları hile yapsa bile, kalan şahısların ellerindeki sırlarla adaleti sağlayabiliyoruz. Ortaya koyduğumuz protokol, tek güvenilir üçüncü şahıs bulunan önceki protokole kıyasla (beklendiği üzere) daha yavaş çalışmaktadır, ancak bu alandaki ilk çok kişilik ve birden fazla güvenilir "iyimser" üçüncü şahıs bulunan adil takas protokolü olma özelliğini taşımaktadır.

ACKNOWLEDGMENTS

I want to use this space to show my gratitude to all the people who made this thesis to be completed. First, I would like to thank to my advisor, Assoc. Prof. Alptekin Küpçü, for his support, understanding, patience and guidance throughout my graduate studies. Then, I would like to thank Dr. Handan Kılınç Alper for her guidance and support.

I want to thank to the thesis commitee, Asst. Prof. Mehmet Emre Gürsoy and Asst. Prof. Mehmet Tahir Sandıkkaya for providing their feedback and support.

I also acknowledge the support of TÜBİTAK, the Scientific and Technological Research Council of Turkey, project 119E088.

I also want to thank to my manager and coworkers at Microsoft to support me and provide learning time for my studies.

Finally, I want to thank to my family, Fatma Nursel Akgül and Mehmet Akgül, to encourage to finish my master degree. I also want to thank to my close friends, Ayberk Sorgun, Pınar Süngü and Can Hakan Dağıdır, for listening to my complaints and be there for me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ABBREVIATIONS

DMFE	Distributed Multi Party Fair Exchange
MFE	Multi Party Fair Exchange
PVSS	Publicly Verifiable Secret Sharing
VE	Verifiable Encryption
TTP	Thrusted Third Party
PPT	Probabilistic Polynomial Time

Chapter 1

INTRODUCTION

In the context of exchange protocols performed by two or more parties, being fair means that either all parties obtain the items that they desire or none of them obtains anything useful. Fairness is generally guaranteed by a trusted third party (TTP). Initial studies involved a single TTP participating at every step of the protocol [Franklin and Reiter, 1997]. Following "optimistic" studies propose solutions where a single TTP optimistically intervenes only in malicious situations [Asokan et al., 1997, Bao et al., 1999, Khill et al., 2001, González-Deleito and Markowitch, 2001, Ben-David et al., 2008, Küpçü and Lysyanskaya, 2010, Alper and Küpçü, 2021, Kılınç Alper and Küpçü, 2021].

There are studies on decentralizing the responsibilities of the TTP; however, they are generally done for two-party settings [Park et al., 2003, Avoine and Vaudenay, 2004, Srivatsa et al., 2005, Zhao and Qin, 2012, Dziembowski et al., 2018, Eckey et al., 2020, Avizheh et al., 2022, Zhang et al., 2023]. For example, Avoine and Vaudenay proposed a distributed TTP solution for two-party fair exchange protocol [Avoine and Vaudenay, 2004]. The solution only works with the two-party case and there is no extension of the work to the multi-party case.

In the multi-party setting, researchers focused on either how to make it more secure and/or fair [Bao et al., 1999, Khill et al., 2001, González-Deleito and Markowitch, 2001, Ben-David et al., 2008, Alper and Küpçü, 2021, Kılınç Alper and Küpçü, 2021] or how to operate using a blockchain structure [Guo et al., 2019, Zhong et al., 2019, Gao et al., 2019, Payeras-Capella et al., 2019, Mut-Puigserver et al., 2020, Ferrer-Gomila and Hinarejos, 2021]. Blockchain studies are not optimistic. Except the blockchain studies, the multi-party cases usually require a single TTP

to prevent malicious attacks. As the name indicates, the protocols start with "TTP is trusted/honest" assumption; however, it becomes a single point of failure.

To eliminate the risk of the single point of failure in terms of security, we propose a distributed (multiple TTP) solution for the multi-party setting. The aim is the provide a fair and optimistic multi-party multi-TTP exchange protocol even with corrupted subset of TTPs. In this thesis, we extend the previous work for multi-party fair exchange [Alper and Küpçü, 2021], which originally uses a single TTP, to obtain the **first** optimistic multi-party fair exchange protocol with multiple TTPs. As of this thesis is written, there is no published study of multi-party fair exchange solution using multiple TTPs (without blockchains) in the literature.

The previous study [Alper and Küpçü, 2021] uses joint encryption and decryption keys to provide fairness, and the single TTP needs to keep the track of conflicts and store the encrypted decryption keys. As an extension, we distribute secret shares to generate these decryption keys and extract the file from encryption with additional computation overhead of $O(m)$, where m is the number of TTPs in our solution. Since the decryption keys are group elements, we use a publicly verifiable secret sharing scheme that supports sharing of group elements to distribute the decryption keys to the TTPs.

As in [Alper and Küpçü, 2021], we achieve **privacy against TTP** behavior of the TTP not learning anything about the exchanged files and being used TTPs only for fairness. We also achieve the **optimistic TTP** behavior, thus, our TTPs only intervene when there is a conflict. Otherwise, they do not even realize that an exchange took place.

Our protocol provides **fairness for honest parties** even with $n-1$ malicious parties and $t-1$ malicious TTPs, where n is the number of all parties and t is the threshold value of the publicly verifiable secret sharing scheme.

Our protocol can be used with **any exchange topology** (parties can exchange items with any combination of other parties). Compared to the previous work [Alper and Küpçü, 2021], we achieve a novel solution for decentralization of TTP with only the cost of communication with m TTPs instead of a single TTP. In our protocol, *broadcasting* is not needed. Our solution and its security proof constitute a non-

trivial extension of the previous work, achieving an important result in this field.

The organization of the thesis as follows. In Chapter 2, the related work on optimistic studies, two party setting and multi party setting and decentralization procedures are summarized. In Chapter 3, the definitions to understand the protocol, notation and a small table of commonly used symbols are provided. Then in Chapter 4, the protocol steps and resolution algorithms are explained in detail. In Chapter 6, the fairness proof is given. Lastly, in Chapter 7, the protocol with the algorithms and calculations of how it can be used is shown in detail and then in Chapter 8, performance numbers for the given instantiation is provided.

Chapter 2

RELATED WORKS

It is known that fair exchange cannot be done without a TTP [Pagnia et al., 1999].

In the early studies, TTP is involved at every step of the protocol [Franklin and Reiter, 1997]. However, it is not effective, since this increases the computational complexity, thus reduces the performance.

2.1 Optimistic Studies

In optimistic studies, TTP intervenes only in case of a conflict [Bao et al., 1998, Bao et al., 1999, González-Deleito and Markowitch, 2001, Küpçü and Lysyanskaya, 2010, Zhao and Qin, 2012, Alper and Küpçü, 2021, Kılınç Alper and Küpçü, 2021]. The purpose is to reduce the communication overhead with the TTP causing bottleneck and enhance the performance. These studies directly focus on the optimistic fair exchange protocols to improve the performance, provide different methods or features. Apart from these studies, there are other studies which uses optimistic fair exchange protocols to solve different problems in another area. For example, [Sandıkkaya et al., 2016] uses optimistic fair exchange protocol to define proposed logging system between the service provider and the user. In their logging system, bulletin board acts as a single TTP of the optimistic two-party fair exchange protocol between service provider and the user when there is a conflict and acts as the write-only tools for communication otherwise [Sandıkkaya et al., 2016].

2.2 Decentralization

Another problem with having one TTP is creating a single point of failure. There are studies focusing on how to decentralize two party settings [Park et al., 2003,

Avoine and Vaudenay, 2004, Srivatsa et al., 2005, Küpçü, 2013, Zhao and Qin, 2012, Dziembowski et al., 2018, Eckey et al., 2020, Avizheh et al., 2022, Zhang et al., 2023]. One way of achieving this is distributing the responsibility of TTP to multiple TTPs, such as [Avoine and Vaudenay, 2004, Zhao and Qin, 2012, Zhang et al., 2023] using publicly verifiable secret sharing (PVSS). Another way is replacing TTP with blockchain [Dziembowski et al., 2018, Eckey et al., 2020, Avizheh et al., 2022].

2.2.1 Two-Party Setting

Avoine and Vaudenay [Avoine and Vaudenay, 2004] provide a solution for two-party setting using PVSS. Instead of one TTP, there are m TTPs to intervene in case of a conflict. In the protocol, the first party sends encrypted secret shares as commitment (can be decrypted by TTPs) to second party and waits for the item. If the commitments are valid, the second party sends its item to first party. If the first party does not receive the item of the second party or the verification fails, it does not send anything and waits for timeout. If the second party does not receive the desired item or the verification fails, it runs recover with TTPs by giving the commitment along with the encryption of its item (can by decrypted by the first party) TTPs verifies the commitments and sends the secret shares to seconds party to recover the item and sends encrypted item to first party.

[Zhang et al., 2023] is a recent published work on decentralized two-party computation using publicly verifiable secret sharing. The main difference is, this new paper adopts decentralized ciphertext-policy attribute-based encryption (CP-ABE) and reduces the complexity of reconstruction.

2.2.2 Multi-Party Setting

Although the two-party solutions are mostly studied, there are studies for the multi-party setting as well [Bao et al., 1999, González-Deleito and Markowitch, 2001, Khill et al., 2001, Ben-David et al., 2008, Alper and Küpçü, 2021, Kılınç Alper and Küpçü, 2021]. These studies are focused on providing secure and fair multi

party exchange and use single TTP to provide optimistic fairness. On the other hand, many [Guo et al., 2019, Zhong et al., 2019, Gao et al., 2019, Payeras-Capella et al., 2019, Mut-Puigserver et al., 2020, Ferrer-Gomila and Hinarejos, 2021] are focused on decentralizing multi-party setting, where blockchain replaces the single TTP. However, there is no solution for optimistic multi party setting using multiple TTPs. Table 2.1 summarizes the most related work.

Paper	**Optimistic**	**#-Party**	**#-TTP**	**Procedure**
[Franklin and Reiter, 1997]	No	2	Single	-
[Avoine and Vaudenay, 2004]	Yes	2	m	Secret Sharing
[Zhao and Qin, 2012]	Yes	2	m	Secret Sharing
[Alper and Küpçü, 2021]	Yes	n	Single	-
[Bao et al., 1999]	Yes	n	Single	-
[Guo et al., 2019]	-	n	None	Blockchain
[Mut-Puigserver et al., 2020]	-	n	None	Blockchain
Ours	Yes	n	m	Secret Sharing

Table 2.1: Comparison of the most related works.

2.3 Our Contribution

Our contribution is proposing the **first** decentralized optimistic multi party fair exchange protocol. We use verifiable secret sharing (VSS) and multiple TTPs to obtain distributed solution for Multi-Party Fair Exchange protocol (MFE). Table 2.1 shows the comparison of our study with the related works.

Our protocol is an enhanced version of previous work [Alper and Küpçü, 2021] combined with publicly verifiable secret sharing algorithm similar to the two-party solution of [Avoine and Vaudenay, 2004]. [Alper and Küpçü, 2021] use distributed encryption for the exchanged items, thus the decryption key can be constructed when a party gets all keys from all parties. When there is a conflict and the TTP

resolves every conflict by having each missing decryption share, the TTP can decrypt the shares and sends them to the parties. The topology of the protocol is not restricted and can be anything. With the idea from [Avoine and Vaudenay, 2004], we distribute each decryption share to multiple TTPs, and parties can recover any decryption share that belongs to another party, and can use it to construct the decryption key. In our case, the secret we want to secret share (i.e. decryption share) is a group element. Most of the secret sharing algorithms [Shamir, 1979, Feldman, 1987, Schoenmakers, 1999, Stadler, 2001, Wu and Tseng, 2011] share a field element; however, any publicly verifiable secret sharing algorithm that supports sharing of group elements could be used (we employ Scrape from [Cascudo and David, 2017]). Our solution and its security proof constitute a non-trivial combination and extension of these closely-related previous studies [Alper and Küpçü, 2021, Avoine and Vaudenay, 2004], where we do not resort to simply using a blockchain structure to achieve multiple TTPs. However, we could still use the blockchain structure, not to achieve multiple TTPs, but as a storage, without losing optimistic behavior. Currently, our TTPs keep separate databases. Blockchain structure can be used as a joint database for TTPs. Since TTPs only write to blockchain if there is a conflict, our protocol preserves to be the first optimistic multi-party multi-TTP fair exchange protocol.

Chapter 3

DEFINITION AND PRELIMINARIES

3.1 Definitions

Definition 3.1.1 (Decisional Diffie-Hellman (DDH) Problem)**.** Let g be the generator of a cyclic group $\mathbb{G}$ with a prime order, let $X, Y, Z \in \mathbb{G}$ and let log be the discrete logarithm, decide whether $Z = g^{\log_g X \log_g Y}$.

Definition 3.1.2 (Zero-Knowledge Proofs of Knowledge of Discrete Logarithm Equality (DLEQ) [Chaum and Pedersen, 1992])**.** The protocol to perform a zero-knowledge proof of knowledge of a value $\alpha \in \mathbb{Z}_q$ s.t. $x = g^\alpha$ and $y = h^\alpha$ given g, x, h, y denoted by by $DLEQ(g, x, h, y)$ works as follows:

1. The prover party computes the following values to send to the verifier party: $a_1 = g^w$ and $a_2 = h^w$ where $w \leftarrow \mathbb{Z}_q$.

2. The verifier party chooses a single challenge $e \leftarrow \mathbb{Z}_q$ to send to the prover party.

3. The prover party computes and sends $z = w - \alpha e$ so that the verifier party can check the if the proof holds.

4. The verifier party checks whether $a_1 = g^z x^e$ and $a_2 = h^z y^e$.

This can be done in full zero knowledge non-interactively in the Random Oracle Model using the Fiat-Shamir heuristic [Amos Fiat, 1988]. We use this method to show the following relation:
$R_{dleq} = \{(\alpha, (\mathbb{G}, p, g, x, h, y | \alpha = \log_g x = \log_h y\}$

DLEQ is used for verifiying the decryption share and the encrypted secret shares.

Definition 3.1.3 (Secure Multi-Party Computation [Alper and Küpçü, 2021])**.** Let π^{ϕ} be a probabilistic polynomial time (PPT) protocol and let ϕ be a PPT multi-party functionality. If for every non-uniform PPT real world adversary $\mathcal{A}$ attacking π $\exists$ a non-uniform PPT ideal world simulator $\mathcal{S}$ for all $f_1, f_2, ..., f_n$ such that

$$\{IDEAL^{s}_{\phi,S(aux),\mathcal{P}_c}(f_1, f_2, ..., f_n, l)_{l\in\mathbb{N}}\} \equiv_c \{REAL^{s}_{\pi,TTP,\mathcal{A}(aux),\mathcal{P}_c}(f_1, f_2, ..., f_n, l)_{l\in\mathbb{N}}\}$$

then, we say that π computes ϕ **securely**.

The honest parties $\mathcal{P}_h$, the malicious parties $\mathcal{P}_c$ corrupted by the adversary $\mathcal{A}$ and the universal trusted party U^{ϕ}_s form the ideal world and the ideal functionality as follows:

U^{ϕ}_s for security with abort [Alper and Küpçü, 2021]

- The honest parties $\mathcal{P}_h$ and the malicious parties $\mathcal{P}_c$ send the $\{f_i\}_{\mathcal{P}_i\in\mathcal{P}_h}$ and $\{f_i\}_{\mathcal{P}_i\in\mathcal{P}_c}$ to U^{ϕ}_s as inputs, respectively.
- If they are invalid or U^{ϕ}_s gets ABORT from $\mathcal{A}$, U^{ϕ}_s sends $\perp$ to all parties.
- If the inputs are valid, then U^{ϕ}_s computes $\phi = (f_1, f_2, ..., f_n) = (\phi_1(f_1, ..., f_n), ..., \phi_n(f_1, ..., f_n))$. Then it sends $\{\phi_i\}_{\mathcal{P}_i\in\mathcal{P}_c}$ to $\mathcal{A}$.
  - If U^{ϕ}_s gets CONTINUE message from $\mathcal{A}$, then it sends $\{\phi_i\}_{\mathcal{P}_i\in\mathcal{P}_h}$ to the honest parties.
  - If U^{ϕ}_s gets ABORT message from $\mathcal{A}$, then it sends $\perp$ to the honest parties.

Definition 3.1.4 (Fair and Secure Multi-Party Computation [Alper and Küpçü, 2021])**.** Let π^{ϕ} be a PPT protocol and let ϕ be a PPT multi-party functionality. If for every non-uniform PPT real world adversary $\mathcal{A}$ attacking π $\exists$ a non-uniform PPT ideal world simulator $\mathcal{S}$ for all $f_1, f_2, ..., f_n$ such that

$$\{IDEAL^{fs}_{\phi,S(aux),\mathcal{P}_c}(f_1, f_2, ..., f_n, l)_{l\in\mathbb{N}}\} \equiv_c \{REAL^{fs}_{\pi,TTP,\mathcal{A}(aux),\mathcal{P}_c}(f_1, f_2, ..., f_n, l)_{l\in\mathbb{N}}\}$$

then, we say that π computes ϕ **fairly and securely**.

The honest parties $\mathcal{P}_h$, the malicious parties $\mathcal{P}_c$ corrupted by the adversary $\mathcal{A}$, the TTPs and the universal trusted party U_{fs}^{ϕ} form the ideal world and the ideal functionality as follow:

U_{fs}^{ϕ} for security and fairness [Alper and Küpçü, 2021]

- The honest parties $\mathcal{P}_h$ and the malicious parties $\mathcal{P}_c$ send the $\{f_i\}_{\mathcal{P}_i \in \mathcal{P}_h}$ and $\{f_i\}_{\mathcal{P}_i \in \mathcal{P}_c}$ to U_{fs}^{ϕ} as inputs, respectively.
- If they are invalid or U_{fs}^{ϕ} gets ABORT from $\mathcal{A}$, U_{fs}^{ϕ} sends $\perp$ to all parties.
- If the inputs are valid, then U_{fs}^{ϕ} computes $\phi = (f_1, f_2, ..., f_n) = (\phi_1(f_1, ..., f_n), ..., \phi_n(f_1, ..., f_n))$. Then it sends $\{\phi_i\}_{\mathcal{P}_i \in \mathcal{P}_c}$ to $\mathcal{A}$.
- Each TTP responds to U_{fs}^{ϕ} with $\{b_i \in \{\mathcal{Y}_i \cup \perp \cup CONTINUE\}\}_{\mathcal{P}_i \in \mathcal{P}_h}$, where $\mathcal{Y}_i$ is the domain of ϕ_i. If TTP is honest, then it always sends $\{b_i = CONTINUE\}_{\mathcal{P}_i \in \mathcal{P}_h}$.
  - If U_{fs}^{ϕ} gets $b_i = CONTINUE$ or $b_i = \phi_i$ for any party $\mathcal{P}_i$ more than the threshold t, then it sends ϕ_i to corresponding $\mathcal{P}_i$.
  - Otherwise, U_{fs}^{ϕ} sends $\perp$ to P_i.

Definition 3.1.5 (Exchange Topology)**.** Let τ_S and τ_R be $n \times n$ binary matrices for sender parties and for receiver parties accordingly. Then, an exchange topology is $\tau = (\tau_S, \tau_R)$. $\tau_S[i, j] = 1$ means that $\mathcal{P}_i$ is sending f_i to $\mathcal{P}_j$ and $\tau_R[i, j] = 1$ means that $\mathcal{P}_i$ is expecting f_j from $\mathcal{P}_j$.

Exchange topology defines which party sends items to which part in the exchange protocol.

Definition 3.1.6 (Fair and Secure Multi-Party Exchange [Alper and Küpçü, 2021])**.** Let π be a PPT protocol for the exchange topology τ and let ϕ be a PPT Multi-

Party Fair exchange (MFE) functionality such that $\phi_i(f_1, ..., f_n) = \{f_j : 1 \leq j \leq n, \tau_S[j, i] = 1\}$. If $\exists$ a PPT simulator $\mathcal{S}$ for every non-uniform PPT adversary $\mathcal{A}$ attacking π and for all $f_1, f_2, ..., f_3$ s.t.

$$\{IDEAL^{fs}_{\phi,S(aux),\mathcal{P}_c}(f_1, f_2, ..., f_n, l)_{l\in\mathbb{N}}\} \equiv_c \{REAL^{fs}_{\pi,TTP,\mathcal{A}(aux),\mathcal{P}_c}(f_1, f_2, ..., f_n, l)_{l\in\mathbb{N}}\}$$

then, we say that π is a fair and secure multi-party exchange with the exchange topology .

Definition 3.1.7 (ψ-Hybrid Model [Canetti, 2000, Alper and Küpçü, 2021])**.** Let π^{ψ} be a PPT protocol for ideal functionality ψ and π^{ϕ} be a PPT protocol for functionality ϕ which uses sub-protocol π^{ψ}. Then ψ-Hybrid Model is using the interaction of parties in π^{phi} with ψ to simplify the security proof of ϕ^{ϕ} instead of executing π^{psi} with each other.

3.2 Preliminaries

Definition 3.2.1 ((k,n)-Threshold Public Key Encryption [Shoup and Gennaro, 2002])**.** It consists of PPT algorithms such as *Key Generation, Encryption, Decryption Share Generation, Decryption Share Proof, Verification, Decryption.*

- Key Generation (ThGen): It generates the joint public key pk, public verification key ν and separate private keys x_i for each party $\mathcal{P}_i$.
- Encryption (ThEnc): It generates ciphertext E from message m using joint public key pk.
- Decryption Share Generation (ThDShare): It generates decryption share d_i from ciphertext E using its private key x_i.
- Decryption Share Proof (ThDSProve): It generates the proof $DSproof_i$ of d_i for ciphertext E.
- Verification (ThDSVerify): It verifies if the given proof $DSproof_i$ is constructed from decryption share d_i for ciphertext E.

- Decryption (ThDec): It returns the message m by decrypting ciphertext E using a set of decryption shares d_i containing at least k shares.

The ideal zero-knowledge functionality between the prover party P who creates decryption share d_p and claims it belongs to ciphertext E_p and the verifier party V who aims to ensure that decryption share d_p derived from ciphertext E_p is as follows:

U^{ZK-R} with a relation R

- The prover party P creates the $proof$ and sends $(proof, id||P||V, w, \delta)$ to U^{ZK-R}
- U^{ZK-R} checks whether $(w, \delta) \in R$ or not. Then, U^{ZK-R} outputs $(valid, id||P||V, \delta)$ or $(invalid, id||P||V, \delta)$.

Definition 3.2.2 (Deniable (k,n)-Threshold Encryption [Alper and Küpçü, 2021])**.** This extends the Definition 3.2.1 adding Deniable Decryption Shares (ThDeny) which generates fake decryption shares, however, the Ideal Zero-Knowledge (stated above) ensures that ThDeny cannot create fake and valid decryption shares in the real world execution, thus it is only used for security proofs.

- Deniable Decryption Share Generation (ThDeny): It generates the fake decryption shares to provide fake message m' from the ciphertext E.

Definition 3.2.3 (Public Key Encryption)**.** It consists of PPT algorithms such as *Key Generation, Encryption, Decryption* using security parameter l.

- Key Generation (PkGen): It generates (pk, sk) pair using security parameter l.
- Encryption (PkEnc): It generates ciptertext VE as the encryption of the given message $m \in \mathbb{M}$ using public key pk (and optionally a label lbl. [Shoup and Gennaro, 2002])

- Decryption (PkDec): It generates the plaintext m as the decryption of given ciphertext VE using secret key sk.

Definition 3.2.4 (Verifiable Public Key Encryption [Camenisch and Damgård, 1998, Camenisch and Damgård, 2000, Camenisch and Shoup, 2003])**.** It extends the Definition 3.2.3 adding verification capability *(Encryption Proof, Encryption Proof Verification)* to the public key encryption scheme without decrypting the ciphertext.

- Encryption Proof (ProveEnc): It generates the ciphertext VE along with its proof $VEproof$, where VE is the output of $PkEnc$ and $VEproof$ is the statement that shows it satifies the predetermined relation R.

- Encryption Proof Verification (VerifyEnc): It decides if the given proof $VEproof$ belongs to VE and satisfies the predetermined relation R.

For the security proof of the verifiable encryption and verification, we use the ideal verifiable encryption between the prover P and the verifier V is as follows:

U^{VE-R} with a relation R

- P sends $(proof, id||P||V, w, \delta, pk, item, lbl)$ to U^{VE-R}.

- U^{VE-R} checks whether $(w, item, \delta) \in R$ or not and whether pk is a valid public key of the encryption scheme or not. If all checks pass, U^{VE-R} outputs $(valid, id||P||V, \delta, pk, lbl, VE)$ to V where $VE = PkEnc(pk, item)$. Otherwise, it outputs $(invalid, id||P||V, \delta, pk, lbl, \perp)$ to V.

Definition 3.2.5 (Publicly Verifiable Secret Sharing (PVSS) [Stadler, 2001])**.** It consists of PPT algorithms such as *Distribution (Deal), Verification (Verify), Reconstruction (Recover).*

- Distribution (PVSSDeal): The dealer computes the shares $s_1, ..., s_n$ for a secret $s \in \mathbb{F}$ for the participants $p_1, ..., p_n$ and encrypts the shares $\hat{s}_i$ using pk_i belongs to participant $\mathcal{P}_i$ along with its proof $Sproof_i$.

- Verification (PVSSVerify): Participants decides if the encryptions $\hat{s}_i$ belongs to the secret s_i verifying $Sproof_i$ interactively or non-interactively.

- Reconstruction (PVSSRecover): Each participant decrypts its share s_i and together with a group of participants, they can reconstruct the secret s using the recover algorithm.

A PVSS must satisfy the following security requirements [Cascudo and David, 2017]:

- **Correctness:** The secret can be reconstructed with the given information when the dealer and the parties are honest.

- **Verifiability:** Verification checks show that the given share is created as a valid share of some secrets and reconstruction checks show that the given share is distributed from the dealer.

- **IND2-Secrecy [Heidarvand and Villar, 2009]:** Any set of t-1 parties (where t is the threshold) does not reveal the secret with the knowledge of the public information and their shares.

Definition 3.2.6 (IND2 Secrecy: Indistinguishability of Secrets [Heidarvand and Villar, 2009])**.** PVSS is IND2-Secret, if any polinomial time adversary $\mathcal{A}$ has an negligible advantage (defined as $|Pr[b = b'] - 1/2|$) by corrupting at most t-1 parties for the following game against challenger $\mathcal{C}$.

1. $\mathcal{C}$ runs Setup of PVSS, creates all public information and (pk, sk) pairs for honest parties. Then $\mathcal{C}$ sends all public keys and public information to $\mathcal{A}$.

2. $\mathcal{A}$ creates (pk, sk) pairs for malicious parties and sends the public keys to $\mathcal{C}$.

3. $\mathcal{A}$ picks x_0 and x_1 from secret space and sends it to $\mathcal{C}$. $\mathcal{C}$ picks $b \longleftarrow \{0, 1\}$ uniformly at random. Then $\mathcal{C}$ runs Deal/Distribution of PVSS for secret x_0 and sends all public information together with x_b to $\mathcal{A}$.

4. $\mathcal{A}$ makes a guess $b' \in \{0, 1\}$.

Definition 3.2.7 (Deniable Publicly Verifiable Secret Sharing)**.** This extends the Definition 3.2.5 adding Deniable Secret Shares (PVSSDeny) which generates fake secret shares. However, PVSSDeny cannot create a set of valid fake secret shares in the real world execution; thus it is only used for security proofs.

- Deniable Secret Share Generation (PVSSDeny): It generates a set of fake secret shares to provide fake decription share d'_i from the ciphertext E.

3.3 Notation and Assumptions

The actively participated parties in the exchange phase are denoted by $\mathcal{P}_i$ where $i = 1, .., n$ and n is the number of actively participated parties. The honest parties are denoted by $\mathcal{P}_h$. We assume the adversary $\mathcal{A}$ corrupts some parties and those malicious parties are denoted by $\mathcal{P}_c$.

The TTPs that replaced single TTP are denoted by $\mathcal{TP}_i$ where $i = 1, ..., m$ and m is the number of TTPs. The honest TTPs are denoted by $\mathcal{TP}_h$. We assume the adversary $\mathcal{A}$ corrupts some TTPs and the malicious TTPs are denoted by $\mathcal{TP}_c$. We assume that the number of the honest TTPs are more than the number of the malicious parties $|\mathcal{TP}_h| > |\mathcal{TP}_c|$. All TTPs are ordered and known by the parties in the same order.

The topology could be anything which is decided at the beginning of the protocol by parties, thus we denote it by a set of parties: $\tau = \{(\mathcal{P}_1, \epsilon_1), ..., (\mathcal{P}_n, \epsilon_n)\}$. ϵ_i contains the sender party and the expected items by $\mathcal{P}_i$ in the form of $(\mathcal{P}_j, F_j)$. For fairness, we need to check if every party has received the expected items from the sender party, thus having "who expects what from who" in the topology is more useful than keeping the track of "who sends what to who". F_j in ϵ_i is the list of the expected items sent by $\mathcal{P}_j$ to $\mathcal{P}_i$.

pk is the jointly generated public key by parties for threshold encryption and ν is the verification key of the threshold encryption. $\{pk_1, ..., pk_m\}$ are the public keys of TTPs $\{\mathcal{TP}_1, ..., \mathcal{TP}_m\}$, known by the parties.

Exchange protocol parameters are decided at Setup phase and defined as $lbl = id||\tau||t_1||t_2||pk||\nu$.

The verifiable encryption is denoted by VE_i indicates that it is sent by $\mathcal{P}_i$ to another party.

$\{VD^k_{i,j}\}$ contains the necessary secret shares for reconstructing the decryption share and it is created for E_i by $\mathcal{P}_j$ and the verifiable encryption can be decrypted by $\mathcal{TP}_k$. t is the threshold of PVSS scheme that is used for creating secret shares. We assume the threshold is between the number of honest TTPs and malicious TTPs $|\mathcal{TP}_h| > t > |\mathcal{TP}_c|$.

We use $d_{i,j}$ for decryption shares created by $\mathcal{P}_j$ for E_i and $s^k_{i,j}$ is the secret share of $d_{i,j}$ created for $\mathcal{TP}_k$.

The following relations are defined for the ideal functionalities $U^{ZK-\mathcal{R}_{ds}}$ and $U^{VE-\mathcal{R}_{item}}$:

$$\mathcal{R}_{ds} = \{(x_i, (pk, d_{i,j}, \nu))|ThDShare(x_i, pk, E) \longrightarrow d_{i,j}\}$$

We also verify the exchanged items by the following relation:

$$\mathcal{R}_{item} = \{(f, (F, c)|F(f, c) \longrightarrow valid\}$$

As an example, $\mathcal{R}_{item}$ could be verifying the signatures of contract signing or verifying the hash of a Bittorent file.

We assume that the adversary cannot block parties talking to TTPs before all resolutions are performed. As in the previous work [Alper and Küpçü, 2021], we assume that parties and TTPs communicate with each other using synchronous network connections. Adversary can only cause a bounded-delay α, which is less than the deadline timeouts. Thus, the protocol is defined in a bounded-delay model [Katz et al., 2013].

Symbols	**Short Description**
n	Number of parties
m	Number of TTPs
t	Threshold of PVSS
τ	Topology
VE_i	Verifiable Encryptions
$\{VD_{i,j}^k\}$	PVSS Escrows
$d_{i,j}$	Decryption share
$s_{i,j}^k$	Secret share
pk	joint public key of threshold encryption
pk_k	public key of $\mathcal{TP}_k$

Table 3.1: Commonly used symbols.

Chapter 4

DISTRIBUTED MULTI PARTY FAIR EXCHANGE

Distributed Multi-Party Fair Exchange (DMFE) is an optimistic fair exchange protocol executed by n parties and m TTPs for any exchange topology. TTPs only participate DMFE optimistically when conflicts arise between parties to preserve fairness. The setup with multiple TTPs decentralizes the TTP concept by distributing the responsibilities among TTPs.

We construct the DMFE protocol by deploying a verifiable and deniable (n, n)-threshold encryption scheme, a verifiable encryption scheme and a publicly verifiable secret share scheme. In a nutshell, DMFE works as follows:

4.1 Overview

The Distributed Multi-Party Exchange has two parts: the main protocol and resolution protocols. The main protocol of DMFE corresponds to the communication between parties to exchange their items. The resolution protocols correspond to the communication between parties and TTPs to preserve fairness by resolving conflicts. If there is no conflict during the protocol, resolutions are not needed and not used.

The main protocol consists of three phases and is visualized in Figure 4.1:

1. Parties communicate with each other to generate the joint public key and their secret key shares for the threshold encryption. This can be done only once among the same set of parties.

2. Parties encrypt their items with the threshold encryption public key. Then, they send the encrypted items to the parties intending to receive them (according to the exchange topology) by providing proof of the correctness of the encrypted item. Now, the protocol reduces to the exchange of the decryption

shares in the complete topology (all pairs of parties) because obtaining all the decryption shares of an encrypted item is the only way to decrypt it (See Definition 3.2.1).

3. Up to now, the main protocol works similarly to the MFE [Alper and Küpçü, 2021]. However, we need a pivotal change due to the assumption of having m TTPs, some of whom might be malicious, as opposed to MFE relying on the existence of a single honest TTP. Therefore, parties in DMFE first generate m encrypted secret shares for each decryption share with PVSS. Each encrypted secret share can be decrypted by only the corresponding TTP. Then they exchange all encryptions of these shares with each other. By deploying PVSS, we aim to prevent malicious parties from collaborating with some TTPs to obtain decryption shares before making sure that honest parties obtain their decryption shares. After receiving correct encryptions, parties send the necessary decryption shares to other parties. When they receive all decryption shares of their desired items, they decrypt and obtain the items.

,

If all parties follow the main protocol honestly, they obtain their desired items in the end. However, if any party deviates from the protocol by not sending the specified data properly, the parties need to engage with TTPs to run the resolution protocols. At the end of resolution protocols, either all parties obtain their desired items or all parties complete the protocol with no useful output (only encrypted items that they cannot decrypt). In short, the resolutions work as follows:

The DMFE has four type of resolution protocol:

1. Resolve 1 is executed when a party does not receive an encrypted secret share of a decryption share. In this case, the party contacts all TTPs and informs the TTPs about this conflict. TTPs responding the party's call, stores the information of this event in a list as the decryption share's secret share is missing. Additionally, the complaining party does not proceed to the next phases in the main protocol.

2. Resolve 2 is executed when a party does not send the decryption shares in the end. If this happens, the party gives all encrypted secret shares that are received to their TTPs. If an encrypted secret share belongs to the decryption share added to a complaint list in Resolve 1, the TTP marks it as resolved. We remark that if an TTP marks it as resolved, it means that the TTP has the secret share of the decryption share which was missing in Resolve 1.

3. If at least one honest TTP resolves all conflicts, all parties can obtain the desired items. To be able to guarantee that, parties execute Resolve 3, during which TTPs give all encrypted secret shares they get in Resolve 2 to the parties. Then, parties merge and give them to other TTPs to resolve remaining conflicts. This way, TTPs can synchronize their complaint list databases, without the need to know or contact each other.

4. After a while, parties executing Resolve 1, 2 and 3 contact TTPs to execute Resolve 4. If an TTP resolved all issues in their list, then they decrypt the encrypted secret share and give the secret share to the requesting party according to the exchange topology. Otherwise (if not all conflicts have been resolved), they do not give any secret share to any requesting party. Due to the synchronization mentioned above, either all honest TTPs have all conflicts resolved and therefore would help all requesting parties, or all honest TTPs have some unresolved conflict remaining and hence would not help any party.

4.2 Protocol

We assume there is a universal order for TTPs, which is known by parties, and that we have more honest TTPs than malicious: $|\mathcal{TP}_h > \mathcal{TP}_c|$. TTPs' public keys $\{pk_k\}$ are known by the parties.

Our protocol consists of the following phases (see Figure 4.1):

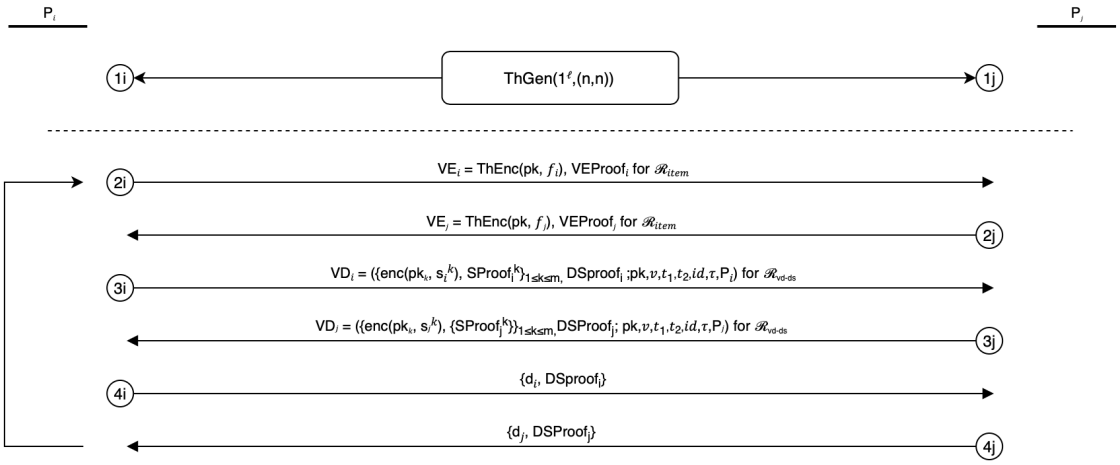

Figure 4.1: Overview of the protocol and the communication between parties.

4.2.1 Setup Phase

Setup is executed *one time* for the given group of parties. After one time setup, they can exchange items as many times as they want. In this phase, parties run the key generation algorithm $ThGen$ to create the joint public key pk, the verification key ν and their secret keys x_i. The joint key generation can be run in a distributed way [Pedersen, 1991] (see Chapter 7 for the instantiation).

Parties also agree on protocol parameters such as deadline times t_1 and t_2 where $t_1 < t_2$, the topology τ of the exchange, and a unique random identification id for parties and TTPs $\{\mathcal{P}_1, ..., \mathcal{P}_n, \mathcal{TP}_1, ..., \mathcal{TP}_m\}$ in this phase.

4.2.2 Encrypted Item Exchange

In this phase, parties exchange their encrypted items with the other parties based on the decided topology as in [Alper and Küpçü, 2021]. After the agreement on parameters in the setup phase, each party $\mathcal{P}_i$ performs the steps below:

- Creates verifiable encryption $VE_i = ThEnc(pk, f_i)$, and its proof $VEproof_i = ProveEnc(pk, (f_i, c_i))$ of the item f_i. Then $\mathcal{P}_i$ sends $VEproof_i$ and VE_i to

the parties which expect the items from $\mathcal{P}_i$ according to topology τ.

- After sending its encrypted item, $\mathcal{P}_i$ waits for the verifiable encrypted items from other parties and verifies them by checking $VerifyEnc(pk, c_j, VE_j, VEproof_j)$ from $\mathcal{P}_j$. If the party does not receive the encrypted items or the encrypted item is invalid, then $\mathcal{P}_i$ aborts locally and does not perform the next phase.

To be able to move forward from this phase, all parties should have the desired encrypted items. To obtain the actual items by decrypting the encrypted items, decryption shares are needed. If any party locally aborts, none of the parties will have all decryption shares at the end of the execution. Therefore, the protocol aborts.

4.2.3 Decryption Share Exchange

Decryption share exchange consists of two parts: sharing the encrypted secret shares and sharing the decryption shares itself. The encrypted secret shares are created by a publicly verifiable secret sharing (PVSS) scheme. In case of an invalid or unreceived value in each part, parties complain to TTPs within the deadline times t_1 and t_2, respectively.

- $\mathcal{P}_i$ creates its decryption share $d_{u,i} \longleftarrow ThDShares(x_i, pk, VE_u)$ for the threshold encryption, and its proof $DSProof_{u,i} \longleftarrow ThDSProve(x_i, pk, d_{u,i}, VE_u)$ for each other party $\mathcal{P}_u$, where $1 \leq i \neq u \leq n$.

- Since we have multiple TTPs, instead of creating verifiable escrows for decryption shares as in [Alper and Küpçü, 2021], $\mathcal{P}_i$ generates m encrypted secret shares (distributed decryption shares) for each decryption share and their proofs i.e., $\{enc(s^k_{u,i}), Sproof^k_{u,i}\} \longleftarrow PVSSDeal(d_{u,i})$, where $1 \leq k \leq m$ and $1 \leq u \neq i \leq n$ using a PVSS scheme. $\mathcal{P}_i$ adds all to the encrypted secret shares list.

- $\mathcal{P}_i$ sends the set of encrypted secret shares (encrypted by TTPs' public keys) and their proofs, associated with a label $id||\tau||t_1||t_2||pk||\nu||\mathcal{P}_j$. When parties receives them, they verify whether the secret shares are valid using $PVSSVerify(enc(s^k_{u,i}), Sproof^k_{u,i})$. Similarly, $\mathcal{P}_i$ expects these encrypted secret shares and their proofs from all other parties.

- If any of the received values is invalid or $\mathcal{P}_i$ does not receive any of them, $\mathcal{P}_i$ executes **Resolve 1** with all TTPs to complain about the misbehaving parties before t_1.

- The parties, which does not have any conflict (that they received all encrypted secret shares and their valid proofs), send the encrypted secret share list with $lbl = id||\tau||t_1||t_2||pk||\nu||\mathcal{P}_j$ to other parties. If verification fails for decryption share or the party does not receive at least one of them, it execute **Resolve 2** with TTPs between t_1 and t_2.

- When a party gets all the shares, it can decrypt the items using $ThDec(\{d_i\}_{1\leq i\leq n}, E)$.

4.3 Resolutions

Each TTP keeps a list `DB` for each conflicting exchanges to resolve. `DB` keeps different information as follows:

- Each TTP keeps track of the conflicts with a list of complaints `complaintList`

- Each TTP keeps a list of verifiable encryptions VE in ϵ.

- Each TTP keeps a table of size $n \times n$ for keep track of the conflicts and whether it is resolved and store secret shares in `DS` for each exchange. All values in `DS` are initially 1 meaning that there is no conflict to resolve. When there is a conflict between two parties $\mathcal{P}_i$ and $\mathcal{P}_j$, the related entry ij of `DS` changes to $\perp$ to indicate some parties did not receive the necessary decryption share. When the conflict is resolved, it changes to the corresponding secret shares.

Each exchange is identified by the followings which are agreed at the setup phase:

- The unique random protocol identifier id.
- The topology τ.
- The deadline times (t_1,t_2).
- The public and verification keys (pk, v) of the threshold encryption.

Overall, we can call them $lbl = id||\tau||t_1||t_2||pk||\nu$.

4.3.1 Resolve 1

Resolve 1 should be executed before t_1. This resolution protocol is executed only to recording misbehaving parties which did not send the set of encrypted secret shares at the first part of the decryption share exchange phase. The parties which do not receive the valid set of encrypted secret shares complain to TTPs. They wait for exchange to be aborted or to be completed with learning secret shares.

When a party $\mathcal{P}_i$ complains about $\mathcal{P}_j$ in Resolve 1, it sends the exchange identifier parameters lbl along with $\{VE_t\}_{t:\tau_S[t,i]=1}$, and complainee party $\mathcal{P}_j$. If $\mathcal{P}_i$ comes after t_1 or the verifiable encryptions are invalid, TTPs do not record the complaint and abort (see lines 1 and 2 in Algorithm 1). If $\mathcal{P}_i$ is the first party coming for Resolve 1, the TTP creates `complaintList`, ϵ and `DS` (lines 4-8). Then it records the complaint in `DB` and asks $\mathcal{P}_i$ to come after t_1 for **Resolve 2** (lines 9-13). The related entry in `DS` of the decryption share for $\mathcal{P}_i$ from $\mathcal{P}_j$ is marked with $\perp$ (line 11).

The identifier parameters specify the `DB` entry. Sending a different parameter will create and return a completely different `DB`.

This algorithm is the same as base protocol [Alper and Küpçü, 2021], since there is no difference while complaining to TTPs. Only difference in our protocol from [Alper and Küpçü, 2021] is that this algorithm runs for multiple times rather than once for complainee, because each TTP performs this independently.

Algorithm 1 Resolve 1

$\mathcal{P}_i$ who complains about $\mathcal{P}_j$ not sending secret shares for $\{VE_t\}_{t:\tau_S[t,i]=1}$, send $\{VE_t\}_{t:\tau_S[t,i]=1}$ with parameters *lbl* and $\mathcal{P}_j$ to each TTP and each TTP does the following if the time t_{now} is before t_1.

1: **if** $t_{now} > t_1$ **or** $t : VerifyEnc(pk, c_t, VE_t, VEProof) \rightarrow invalid$ **then**
2: **return** ABORT
3: **else**
4: **if** $DB[lbl]$ **does not exist then**
5: $\texttt{complaintList}, \epsilon \leftarrow \varnothing$
6: $\{\texttt{DS}[i,j] \leftarrow 1\}_{1 \leq i,j \leq n}$
7: $\texttt{DB}[lbl] \coloneqq \texttt{complaintList}, \epsilon, \texttt{DS}$
8: **end if**
9: $\{\epsilon[t] \coloneqq VE_t\}_{t:\tau_S[t,i]=1}$
10: get $\texttt{complaintList} \in \texttt{DB}[lbl]$
11: $\texttt{DS}[t,j] \coloneqq \perp$ for all t such that $\tau_S[t,i] = 1$
12: $\texttt{complaintList} \coloneqq \texttt{complaintList} \cup \{\mathcal{P}_i, \mathcal{P}_j\}$
13: **return** "Come after t_1 for **Resolve 2**"
14: **end if**

4.3.2 Resolve 2

Resolve 2 is executed between t_1 and t_2. In this resolution, TTPs restore the missing secret shares for the complaint parties.

When a party $\mathcal{P}_i$ proceeds with Resolve 2 within the time period between t_1 and t_2, the TTP $\mathcal{TP}_k$ first checks if there is any conflict recorded in DS (see line 3 of Algorithm 2). If there is no conflict, it returns "Come after t_2 for Resolve 3" (line 22). If there is a conflict, $\mathcal{TP}_k$ requests the encrypted secret shares (lines 3-4). If $\mathcal{P}_i$ does not send the encrypted secret shares, $\mathcal{TP}_k$ returns "Perform Resolve 3" (lines 5-6). When $\mathcal{TP}_k$ receives encrypted secret share, it checks whether they are valid and changes corresponding DS entries from $\perp$s to given encrypted secret shares (lines 8-15).

Algorithm 2 Resolve 2

$\mathcal{P}_i$ comes with parameters *lbl* to each $\mathcal{TP}_k$. $\mathcal{TP}_k$ does the following:

if $t_2 > t_{now} > t_1$ **then**
  $\texttt{complaintList}, \texttt{DS} \leftarrow \texttt{DB}[lbl]$
  if there exists j such that $\texttt{DS}[i,j] = \perp$ **then**
    request $\{VD^k_{i,j} : \exists u \text{ s.t. } \texttt{DS}[u,j] = \perp, \tau_s[u,i] = 1\}$
    if $\mathcal{P}_i$ does not send $VD^k_{i,j}$ before t_2 **then**
      return "Perform **Resolve 3**"
    end if
    $D = D \cup \{VD^k_{i,j}, \{enc(pk_k, s^k_{i,j})\}\}$
    for all $\{VD^k_{i,t}, \{enc(pk_k, s^k_{i,t})\}\} \in D$ **do**
      $VE_t = \epsilon[t]$
      if $Verify(VD^k_{i,j}) \rightarrow invalid$ **then**
        return FAIL
      end if
      $\texttt{DS}[t,i] := \{\text{VD}^k_{i,t}, \{enc(pk_k, s^k_{i,t})\}\}$
    end for
    if $\{DS[x,y] \neq \perp\}_{1 \leq x,y \leq n}$ **then**
      return "Perform **Resolve 3**"
    else
      return "Come after t_2 for **Resolve 3**"
    end if
  else
    return "Come after t_2 for **Resolve 3**"
  end if
end if

The difference in Resolve 2 from [Alper and Küpçü, 2021] is, in the protocol we do not decrypt the given encrypted secret shares yet. We store them and decrypt them in Resolve 4.

4.3.3 Resolve 3

This resolution is needed to preserve fairness when a malicious party resolves conflicts with few honest TTPs only (less than threshold-many) and cooperates with malicious TTPs to reach the threshold and reconstruct the decryption shares. Honest parties communicate with each TTP after t_2. Parties request the encrypted secret shares given in **Resolve 2** from TTPs. Parties merge all collect encrypted secret shares and send them to all TTPs in **Resolve 3** (Algorithm 3). Therefore, all honest TTPs have the same set of encrypted secret shares.

Algorithm 3 Resolve 3

$\mathcal{P}_i$ comes with parameters *lbl* that TTPs send in Resolve 2.

1: **if** $\mathrm{t}_{now} > t_2$ **then**
2: `DS` $\leftarrow$ `DB`[*lbl*]
3: **return** `DS`
4: **end if**

4.3.4 Resolve 4

This resolution either outputs "ABORTED" or gives necessary shares to the parties. If `DS` still has $\perp$ after t_2, parties might give the encrypted secret shares obtained from other TTPs in **Resolve 3** to resolve conflicts.

First, $\mathcal{TP}_k$ checks if there is any recorded conflict in `DS`. If there is a conflict, $\mathcal{TP}_k$ requests the encrypted secret shares (see lines 5-6 of Algorithm 4). If $\mathcal{TP}_k$ receives encrypted secret shares, it checks whether they are valid and changes corresponding `DS` entries from $\perp$s to given encrypted secret shares (lines 7-15). If there is no conflict remaining (or resolves after receiving encrypted secret shares), $\mathcal{TP}_k$ decrypts $VD_{(}^{k}u, i)$ to send $\mathcal{P}_i$ (lines 17-25). Otherwise, it *ABORTS* (line 27).

4.4 Optimization Discussion

Resolution 3 requires additional communication with TTPs to unify their databases. This step is needed to preserve fairness as stated. Note that given a threshold t,

we assume there are at least t honest TTPs, and at most $t-1$ malicious TTPs. Malicious TTPs may only help malicious parties. But malicious parties need help from at least one honest TTP. With the synchronization among TTPs performed by honest parties, they will all have the same database, and therefore all will either have all conflicts resolved, or some conflict remaining. This would ensure fairness even with multiple TTPs. There are other options to optimize the communication. In the current setting, each TTP stores its own database. Therefore, the `complaintList`, resolved conflicts, given encrypted secret shares etc. might be different for each TTP. In this thesis, we unify these values by downloading and sending in resolution steps. However, it can be done by defining a joint database or storing them in a blockchain. Using blockchain as storage does not change the optimistic behavior of the protocol. TTPs only write to the blockchain when there is a conflict. Moreover, TTPs do not need a private blockchain. Since the secret shares are encrypted, public blockchain can be used for storage.

Algorithm 4 Resolve 4

$\mathcal{P}_i$ comes with parameters lbl.

if $t_2 > t_{now}$ **then**
  return "Try after t_2"
end if
$\texttt{DS} \leftarrow \texttt{DB}[lbl]$
if there exists j such that $\texttt{DS}[i,j] = \perp$ **then**
  request $\{VD_{i,j} : \exists u \text{ s.t. } \texttt{DS}[u,j] = \perp, \tau_s[u,i] = 1\}$
  if $\mathcal{P}_i$ sends $VD^k_{i,j}$ **then**
    $D = D \cup \{VD_{i,j}, \{enc(pk_k, s^k_{i,j})\}\}$
    for all $\{VD_{i,t}, \{enc(pk_k, s^k_{i,t})\}\} \in D$ **do**
      $VE_t = \epsilon[t]$
      if $Verify(VD_{i,j}) \rightarrow valid$ **then**
        $\texttt{DS}[t,i] := \{\text{VD}_{i,t}, \{enc(pk_k, s^k_{i,t})\}\}$
      end if
    end for
  end if
end if
if $\{DS[x,y] \neq \perp\}_{1 \leq x,y \leq n}$ **then**
  $D_i := \varnothing$
  $D_i := D_i \cup \texttt{DS}[t,j]$ **for all** $\mathcal{P}_j, t : (\mathcal{P}_i, \mathcal{P}_j) \in \texttt{complaintList}$ **and** $\tau_S[t,i] = 1$
  if $\mathcal{V} \neq \varnothing$ **then**
    for all $VD^k_{u,i} \in \mathcal{V}$ such that $lbl = id||\tau||t_1||t_2||pk||\nu||\mathcal{P}_j$ **do**
      $D_i := D_i \cup \{VD^k_{u,i}, \{PkDec(sk_k), enc(pk_k, s^k_{u,i})\}\}$
    end for
  end if
  return D_i
else
  return ABORT
end if

Chapter 5

EXAMPLE EXECUTION WITH CONFLICT

Assume that we have 3 parties to exchange items and 5 TTPs to interact in case of a conflict. The threshold of PVSS is 3, so any party needs at least 3 secret shares to reconstruct the decryption share. The exchange topology is complete, so each party sends items to all parties. $\mathcal{P}_1$ and $\mathcal{P}_2$ are honest parties and $\mathcal{P}_3$ is malicious party. $\mathcal{TP}_1$, $\mathcal{TP}_2$, $\mathcal{TP}_3$ are honest TTPs and $\mathcal{TP}_4$, $\mathcal{TP}_5$ are malicious TTPs, cooperates with $\mathcal{P}_3$.

First, they send encrypted items VE_i and encrypted secret shares VD_i.

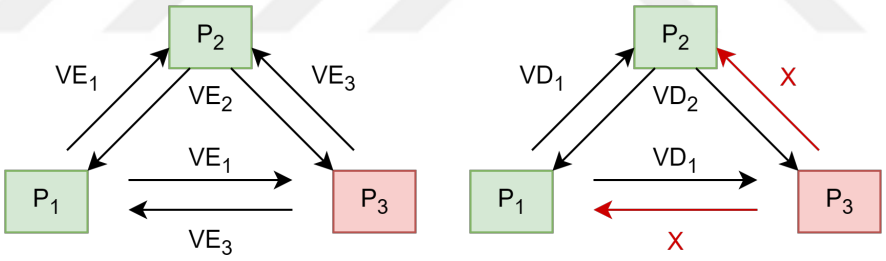

Figure 5.1: Step 2 and Step 3 of the Main Protocol where $\mathcal{P}_3$ does not send VD_3

$\mathcal{P}_3$ wants to obtain the items without giving hers/his item, however, $\mathcal{P}_3$ needs to share VE_3 so that other parties continues to share the encrypted secret shares. $\mathcal{P}_3$ waits for VD_1 and VD_2, but does not send VD_3 to other parties. In Figure 5.1 demonstrates the communication between the parties.

$\mathcal{P}_1$ and $\mathcal{P}_2$ wait for VD_3 in step 3 and go to TTPs to complain about $\mathcal{P}_3$ as in Figure 5.2. $\mathcal{P}_1$ and $\mathcal{P}_2$ do not continue with the protocol and do not send decryption shares to $\mathcal{P}_3$.

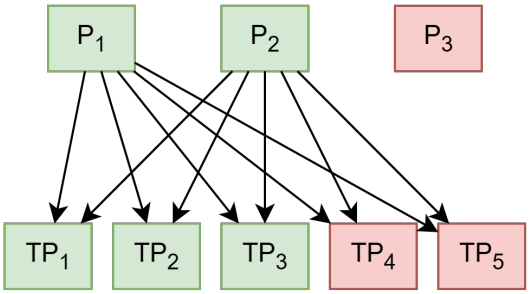

Figure 5.2: $\mathcal{P}_1$ and $\mathcal{P}_2$ complain about $\mathcal{P}_3$ not sending VD_3 to all TTPs.

After the deadline is passed, $\mathcal{P}_1$ and $\mathcal{P}_2$ performs Resolve 2 with all TTPs, however, $\mathcal{P}_3$ performs Resolve 2 with only malicious parties and with only 1 honest party, $\mathcal{TP}_3$, as in Figure 5.3. Now, all TTPs has VD_1 and VD_2. If there is no conflict in at least 3 TTPs, $\mathcal{P}_3$ can reconstruct the decryption shares getting them from TTPs. On the other hand, only $\mathcal{TP}_3$, $\mathcal{TP}_4$ and $\mathcal{TP}_5$ have VD_3 and conflict is resolved only in these TTPs. At the end of Resolve 2, $\mathcal{TP}_1$ and $\mathcal{TP}_2$ still have unresolved conflict.

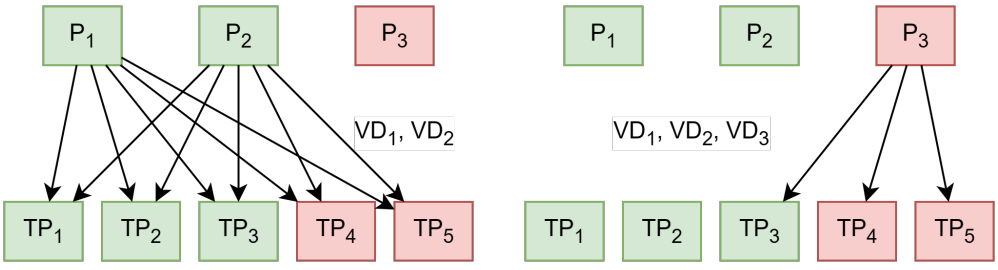

Figure 5.3: Parties perform Resolve 2.

In the base protocol [Alper and Küpçü, 2021], TTP sends the decryption shares in Resolve 3 if the all conflicts are resolved. If we use the same Resolve 3, $\mathcal{TP}_3$ would share the decrypted secret shares of VD_1, VD_2 and VD_3 with all parties,

but malicious TTPs $\mathcal{TP}_4$ and $\mathcal{TP}_5$ would share the decryption of VD_1 and VD_2 with only $\mathcal{P}_3$. $\mathcal{P}_3$ would be able to reconstruct the decryption shares by getting 3 decrypted secret shares, however, $\mathcal{P}_1$ and $\mathcal{P}_2$ would not be able to reconstruct since they only have 1 decrypted secret shares coming from $\mathcal{TP}_3$. Therefore, $\mathcal{P}_3$ would be able to obtain items where $\mathcal{P}_1$ and $\mathcal{P}_2$ abort the protocol. Fairness is broken.

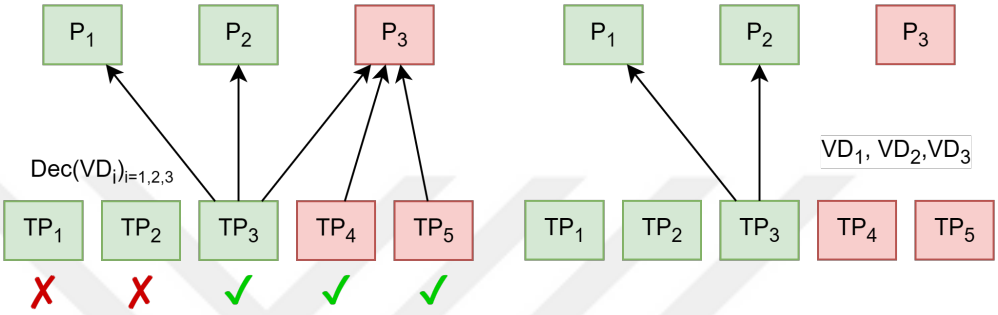

Figure 5.4: Old resolve 3 vs New resolve 3

To prevent this attack, we added new step (as our new Resolve 3) before decrypting the secret shares. Parties interact with TTPs to collect encrypted secret shares that they obtained in Resolve 2. The aim is if any honest party resolves all conflict, we need to guarantee that all honest parties can resolve all conflicts. $\mathcal{TP}_3$ sends encrypted VD_1, VD_2 and VD_3 to $\mathcal{P}_1$ and $\mathcal{P}_2$ when they perform Resolve 3. Figure 5.4 shows the difference of the communication with TTPs in the base protocol [Alper and Küpçü, 2021] Resolve 3 and our new Resolve 3.

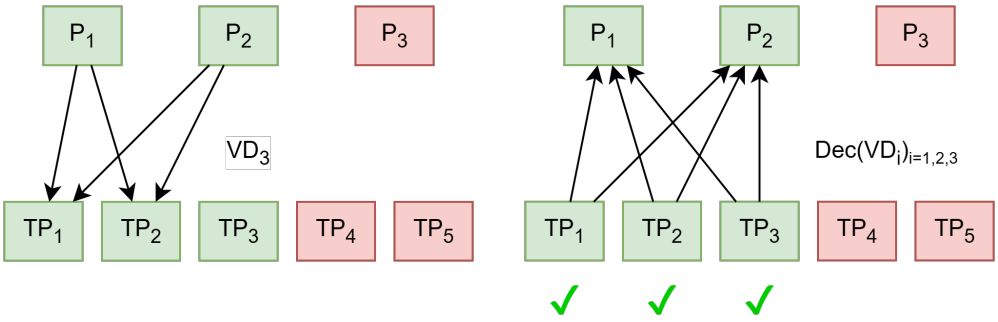

Figure 5.5: Parties perform resolve 4.

In Resolve 4 shown in Figure 5.5, $\mathcal{P}_1$ and $\mathcal{P}_2$ send VD_3 to $\mathcal{TP}_1$ and $\mathcal{TP}_2$ that they collected in Resolve 3 from $\mathcal{TP}_3$. Now, $\mathcal{TP}_1$ and $\mathcal{TP}_2$ can resolve the conflicts as $\mathcal{TP}_3$ and send decrypted VD_3 to $\mathcal{P}_1$ and $\mathcal{P}_2$. All honest parties can reconstruct the decryption shares and fairness is preserved.

Chapter 6

FAIRNESS PROOF

Theorem. *The Distributed Multi-Party Exchange protocol with the topology τ realizes U_{fs}^{ϕ} in the $U^{VE-R_{item}}$, $U^{VD-R_{vs-ds}}$, $U^{ZK-R_{ds}}$ hybrid model, where ϕ in U_{fs}^{ϕ} defined in Definition 3.1, under the following assumptions:*

1. $|\mathcal{TP}_h| > |\mathcal{TP}_c|$
2. ***IND-CPA*** *is guaranteed for the verifiable deniable (n, n)-threshold encryption scheme which is used for creating verifiable encryptions (VE) (inherited from [Alper and Küpçü, 2021]).*
3. ***IND2-Secrecy*** *is guaranteed for the publicly verifiable secret sharing (PVSS) scheme.*

Proof. As stated in Theorem 6, we prove the fairness in the $U^{VE-R_{item}}$, $U^{VD-R_{vs-ds}}$ and $U^{ZK-R_{ds}}$ hybrid model. Let $\mathcal{P}_c$ and $\mathcal{TP}_c$ be the malicious parties and TTPs, respectively, corrupted by $\mathcal{A}$, $\mathcal{P}_h$ and $\mathcal{TP}_h$ be the honest parties and TTPs, respectively, simulated by $\mathcal{S}$ as defined in Definition 3.1. In more detail, $\mathcal{S}$ simulates the parties in $\mathcal{P}_h$ and $\mathcal{TP}_h$ in the real world, whereas simulates parties in $\mathcal{P}_c$ and $\mathcal{TP}_c$ in the ideal world. The simulation works as follows:

Setup

- $\mathcal{S}$ generates the set of public key-private key pairs (pk_m, sk_m) for each TTP in $\mathcal{TP}_h$ and publishes the public keys in the real world.
- Then, $\mathcal{S}$ generates the joint public key pk, their secret keys $x_1, ..., x_n$ and the verification key ν running as $ThGen$.

- Finally, $\mathcal{S}$ sends the secret keys of the malicious parties in $\mathcal{P}_c$ together with pk and ν.

Encrypted Item Exchange

- $\mathcal{S}$ simulates U^{VE-R} as described in Definition 3.2.

- $\mathcal{S}$ does not know the real items of honest parties at this point. Therefore, it simulates the honest parties by sending the encryption of random items to malicious parties instead of real items. We note that $\mathcal{S}$ can send verifiable encryption of random items for the relation R because $\mathcal{S}$ is $U^{VE-\mathcal{R}}$.

- If $\mathcal{S}$, as $U^{VE-\mathcal{R}}$, receives the items f_j from adversary $\mathcal{A}$ sent to an honest party, $\mathcal{S}$ learns the item of malicious party P_j if it is valid.

In the end of this phase, $\mathcal{S}$ can be in the following cases:

Case 1. $\mathcal{S}$ did not receive all items of malicious parties

If $\mathcal{S}$ is in this case, $\mathcal{S}$ aborts the protocol for honest parties that did not receive verifiable encryption of some malicious parties. Thus $\mathcal{S}$ does not proceed with the decryption share exchange phase for these honest parties. We now explain the simulation of honest parties in this case in more detail:

We let $\mathcal{P}_{1h}$ be the honest parties that do not get some of the encrypted items and $\mathcal{P}_{2h}$ be the honest parties that get the encrypted items of all malicious parties. $\mathcal{S}$ performs step 3 for parties in $\mathcal{P}_{2h}$ parties and do not proceed for parties in $\mathcal{P}_{1h}$. Since $\mathcal{S}$ aborted for parties in $\mathcal{P}_{1h}$, the parties in $\mathcal{P}_{2h}$ and malicious parties do not receive encrypted secret shares in step 3. Therefore, $\mathcal{S}$ performs Resolve 1 with TTPs including malicious TTPs as described in the protocol on behalf of parties in $\mathcal{P}_{2h}$. $\mathcal{S}$ performs Resolve 1 for malicious parties if they do not execute step 3 properly. After t_1, t_2 and t_3, $\mathcal{S}$ performs Resolve 2, Resolve 3 and Resolve 4, respectively. During the simulation of the honest TTPs, conflicts

cannot be resolved since the decryption share and secret shares are never created for $\mathcal{P}_{1h}$. Since the protocol uses (n,n)-threshold encryption, parties cannot decrypt the items. $\mathcal{S}$ sends $ABORT$ to U_{fs}^{ϕ} and completes the simulation.

In the real-world execution, honest parties would go the TTPs to complain about the missing secret shares that $\{\mathcal{P}_{1h}\}$ do not send. Neither the TTPs nor the honest parties would have the secret shares of the aborted parties. Thus, none of the parties (malicious or honest) would obtain the items.

If all parties receive the encrypted items, they continue with Decryption Share Exchange.

Case 2. $\mathcal{S}$ does not have all encrypted secret shares.

This case does not guarantee that honest parties will obtain items at the end of the execution. We note that, in this case, all honest parties have verifiable encryptions of malicious parties' items. Therefore, $\mathcal{S}$ needs to simulate step 3 on behalf of honest parties. $\mathcal{S}$ simulates the step 3 as described in the protocol. Even though the shared decryption shares of honest parties in this step do not decrypt the correct items of honest parties, it is not an issue at this point. It will be handled at the point when it is guaranteed that all parties obtain their desired items.

We let $\mathcal{P}_{3h}$ be the honest parties that do not get some encrypted secret shares of malicious parties and $\mathcal{P}_{4h}$ be the ones that completes the step 3 with all malicious parties correctly. $\mathcal{S}$ performs step 4 for parties in $\mathcal{P}_{4h}$ as described in the protocol and do not proceed to step 4 for $\mathcal{P}_{3h}$. Thus, $\mathcal{S}$ does not send the decryption shares of honest parties in $\mathcal{P}_{3h}$ to malicious parties and it performs Resolve 2 on behalf of them as described in the protocol.

If malicious parties do not send encrypted secret shares to resolve conflicts, $\mathcal{S}$ sends $ABORT$ to U_{fs}^{ϕ}, does not send encrypted secret shares on

behalf of honest TTPs as in the real Resolve 2 protocol and completes the simulation. Since the number of malicious TTPs is less than the threshold, the secret shares collected from malicious TTPs by malicious parties are not enough to recover decryption shares of honest parties.

If the malicious parties send their encrypted secret shares during the resolutions and $\mathcal{S}$ resolves all conflicts for at least t honest TTPs, then $\mathcal{S}$ performs Resolve 43 as in Case 3. In more detail, when $\mathcal{S}$ has at least t decrypted secret shares of all malicious items that are complained in Resolve 2, where t is the threshold of PVSS, $\mathcal{S}$ understands that honest parties will get the items in the real protocol. Therefore, it learns the real items from U_{fs}^{ϕ}. Then it generates the fake secret shares for each TTP having the missing secret shares. For this, it first runs ThDeny to obtain fake decryption shares of parties in $\mathcal{P}_{3h}$ that decrypts the items of honest parties obtained from U_{fs}^{ϕ} together with the given decryption shares by parties in $\mathcal{P}_{3h}$. Then, it runs PVSSDeny to obtain fake secret share for each honest TTP during Resolve 4 so that the reconstruction outputs the fake decryption shares.

Case 3. $\mathcal{S}$ does not have all decryption shares, but have all encrypted secret shares.

This case means that all malicious parties sent their encrypted secret shares to at least one honest party, thus $\mathcal{S}$ has enough secret shares and is able to reconstruct all decryption shares even though the malicious parties do not send their decryption shares.

In the real world execution, honest parties would receive the decryption shares in the second part of decryption share exchange phase or perform the resolutions. Since at least one honest party has the encrypted secret shares of each malicious party, they would share them with TTPs and reconstruct the decryption shares.

One thing to pay attention here is $\mathcal{S}$ has to have at least t honest TTPs to be able to decrypt at least t secret share and reconstruct a decryption

share by itself.

$\mathcal{S}$ decrypts all verifiable encryptions VE_j to get all $\{f_j\}_{\mathcal{P}_j \in \mathcal{P}_c}$. Then $\mathcal{S}$ sends them to U_{fs}^{ϕ} and receives $\{f_i\}_{\mathcal{P}_i \in \mathcal{P}_h}$ as a response. Then, $\mathcal{S}$ runs ThDeny to generate the decryption shares which decrypt the real items. The simulation ends when $\mathcal{S}$ have all valid decryption shares of all malicious parties.

If $\mathcal{S}$ does not receive valid decryption shares of all malicious parties, it performs **Resolve 2** and **Resolve 3** to resolve all conflicts for honest parties with honest TTPs. It also performs **Resolve 2** and **Resolve 3** with malicious parties (though it is not needed to since $\mathcal{S}$ has at least t honest TTPs).

If malicious parties perform the **Resolve 3** and there is no conflict, $\mathcal{S}$ creates fake secret shares to reconstruct a valid decryption share for $\tilde{VE}$s by running $PVSSDeny$ and sends them to malicious parties.

Claim 1 The interaction with the real parties and the simulator $\mathcal{S}$ are indistinguishable for an adversary $\mathcal{A}$.

We prove this claim step by step starting with the real protocol and end with the simulator $\mathcal{S}$.

Game 1. $\mathcal{P}_c$ is the set of malicious parties which are corrupted by the adversary $\mathcal{A}$ and $\mathcal{P}_h$ is the set of honest parties. The simulator $\mathcal{S}$ acts as in MFE in U^{ZK-R} and U^{VE-R}, it takes the items and inputs of all honest parties as inputs to itself and simulates the protocol as honest parties and TTPs. The simulation is indistinguishable from MFE.

Game 2. What we change in the game 2 is the simulator acts as ideal functionalities and takes followings as inputs to evaluate if they are valid when they come from adversary and outputs in behalf of honest parties against the adversary:

- $(proof, id||\mathcal{P}_i||., .)$ as U^{ZK-R}
- $(VEproof, id||\mathcal{P}_i||., ., pk, , VE_i)$ as U^{VE-R}

The game is indistinguishable from Game 1.

Game 3. What we change in the game is simulator will send encrypted random secret shares but not decryption shares, thus corrupted parties perform the resolutions. Since the number of corrupted parties is less than the threshold of PVSS, malicious parties need to get distributed decryption shares from honest TTPs. Simulator records their complaints in Resolve 1 and gets all what they have in Resolve 2. Then it decrypts the needed distributed decryption shares in Resolve 3 and sends them. However, since honest parties sent random distributed decrytion shares, simulator uses PVSSDeny to create fake distributed decryption shares to provide real decryption share using the fake ones that malicious parties already have. From the IND2-Secrecy property of PVSS, having t-1 decryption shares does not reveal any information about the decryption share and it is indistinguishable. We define a hybrid game to show the reduction of the game. Let $r = |\mathcal{P}_h|$ and $H_{3,i}$ is the hybrid game where $\mathcal{P}_h = \{\mathcal{P}_i\}_{1 \leq i \leq r}$ and i is the number of honest parties in Game 2 and others acts as in Game 3. We show that the adversary is not able to distinguish $H_{3,0}$ (Game 3) and $H_{3,r}$ (Game 2), thus it is not able to distinguish $H_{3,i}$ from $H_{3,i+1}$ for some random i. If it is distinguishable, then there can be found a PPT adversary $\mathcal{B}$ that breaks IND2-Secrecy property of PVSS.

The IND2-Secrecy Challenger runs the setup of PVSS and creates all public information, then sends public information and public keys of the honest TTPs to adversary $\mathcal{B}$. The adversary $\mathcal{B}$ creates and sends the public keys of the malicious TTPs back to the challenger.

- $\mathcal{B}$ asks the challenger to deal the decryption shares of the first i parties $\{\mathcal{P}_1, ..., \mathcal{P}_i\}$.

- $\mathcal{B}$ asks the challenger to deal a random element from decryption share space for parties $\{\mathcal{P}_{i+2}, ..., \mathcal{P}_n\}$.

Then $\mathcal{B}$ picks x_0 and x_1 from the decryption share space, where x_0 is the real decryption share for $\mathcal{P}_{i+1}$ and x_1 is random element from the decryption share space. The challenger picks random bit $b \longleftarrow [0, 1]$ and deals x_b. Then, the challenger sends

distributed decryption shares.

The adversary $\mathcal{B}$ corrupts at most $t-1$ TTPs and learns the distributed shares. Assume that the adversary distinguishes if b is 0 or 1 with $adv(l)$ advantage and the probability of guessing i correct is $1/r$, then we can say that the public encryption scheme, which is used in PVSS, is broken by $adv(l)/r$. If the adversary can distinguish, then $adv(l)$ is non-negligible, thus $adv(l)/r$ must be non-negligible, however, it contradicts with IND2-Secrecy property of PVSS. IND2-Secrecy property enforces that the $adv(l)/r$ is negligible, thus $adv(l)$ is negligible. Then, adversary is not able to distinguish Game 3 from Game 2.

Game 4. What we change in the game is the simulator $\mathcal{S}$ will send the random encryption items at the Encrypted Item Exchange and will use ThDeny to create fake decryption shares and deal them to provide fake secret shares. Each $\mathcal{P}_i$ will choose a random item and will encrypt it. Since we do not know which item belongs to which party, the secret shares is be dealed from fake decryption shares created by ThDeny.

We define hybrid games to show the reduction of the games. Let $r = |\mathcal{P}_h|$ and $H_{4,i}$ is the hybrid game where $\mathcal{P}_h = \{\mathcal{P}_i\}_{1 \leq i \leq r}$ and i is the number of honest parties in Game 3 and others acts as in Game 4. We show that the adversary is not able to distinguish $H_{4,0}$ (Game 4) and $H_{4,r}$ (Game 3), thus it is not able to distinguish $H_{4,i}$ from $H_{4,i+1}$ for some random i. To be able to show that, $\mathcal{B}$ plays both against the adversary and against the IND-CPA challenger. $\mathcal{B}$ simulates $\{\mathcal{P}_1, ..., \mathcal{P}_r\}$ without knowing their secret keys as in Game 4 against the adversary and against IND-CPA challenger, $\mathcal{B}$ simulates $\{\mathcal{P}_{r+1}, ..., \mathcal{P}_n\}$ as in Game 3. $\mathcal{B}$ learns the secret keys $\{x_{m+1}, ..., x_n\}$ from IND-CPA challenger along with the joint public key pk and its verification key ν. If adversary is able to distinguish $H_{3,i}$ from $H_{3,i+1}$ for some random i, then there can be found a PPT adversary $\mathcal{B}$ that breaks indistinguishability under chosen plaintext attack (IND-CPA) property of the threshold encryption.

$\mathcal{B}$ picks a random i from $[0, m]$ and simulates the first i parties in Game 3 and from $\mathcal{P}_{i+2}$ to the rest in Game 4. In other words, first i parties encrypt the actual items and from $\mathcal{P}_{i+2}$ to the rest use random items. $\mathcal{P}_{i+1}$ will act randomly as in $H_{4,i+2}$ (Game 4) or as in $H_{4,i}$ (Game 3). As a challenge, $\mathcal{B}$ sends both the actual

and random item for $i+1$ to IND-CPA challenger and gets $\tilde{VE}_{i+1}$ as the encryption without knowing whether it belongs to actual item or not.

$\mathcal{B}$ sends the encryption of actual items for $\{P{-}_1, ..., \mathcal{P}_i\}$, $\tilde{VE}_{i+1}$ for $\mathcal{P}_{i+1}$ and the encryption of random items for $\{\mathcal{P}_{i+2}, ..., \mathcal{P}_m\}$. Then, $\mathcal{B}$ simulates U^{VS-R} to learn the decryption shares from the adversary.

In the decryption share exchange, $\mathcal{B}$ learns the decryption shares from adversary by simulating U^{VS-R} and sends the decryption shares outputted by ThDeny as follows:

- For the first i parties, $\mathcal{B}$ uses the actual item and their encryptions.
- For $i+1$, $\mathcal{B}$ uses the actual item and $\tilde{VE}_{i+1}$. Keep in mind that $\mathcal{B}$ does not know whether $\tilde{VE}_{i+1}$ belongs to the actual item or not. If it is, then $\mathcal{B}$ plays Game 3, if not, $\mathcal{B}$ plays Game 4.
- From $i+2$ to rest, $\mathcal{B}$ uses the random items and random encryptions.

The proofs of the decryption shares are simulated by U^{ZK-R}

Assume that the adversary distinguishes if $\tilde{VE}_{i+1}$ belongs to actual item or not with $adv(l)$ advantage and the probability of guessing i correct is $1/r$. Then by $adv(l)/r$ advantage, we can break the IND-CPA property of the threshold encryption. If the adversary can distinguish the actual item from the random one, then $adv(l)$ is non-negligible, thus $adv(l)/r$ must be non-negligible, however, IND-CPA property enforces that the $adv(l)/r$ is negligible, thus $adv(l)$ is negligible. Then, adversary is not able to distinguish Game 4 from Game 3.

Claim 2 The outputs of honest and malicious parties in the real world and in the ideal world are indistinguishable.

$$\{IDEAL^{fs}_{\phi,S(aux),\mathcal{P}_c}(f_1, f_2, ..., f_n, l)_{l\in\mathbb{N}}\} \equiv_c \{REAL^{fs}_{\pi,TTP,\mathcal{A}(aux),\mathcal{P}_c}(f_1, f_2, ..., f_n, l)_{l\in\mathbb{N}}\}$$

Proof. Above, we proved that the simulation of the honest parties by the simulator $\mathcal{S}$ and the DMFE protocol are indistinguishable. Here, we prove that the outputs of all parties (honest and malicious) are indistinguishable for real and ideal worlds.

Case 1. The protocol is completed such that all parties could obtain their desired items. S acts as the ideal adversary, sending the message $\{f_i\}_{\mathcal{P}_i \in \mathcal{P}_h}$ to represent the ideal scenario, and $b_i = CONTINUE$ as each TTPs to U_{fs}^{ϕ}, ensuring that honest participants receive their expected items. It is shown that the honest parties receives all items $\{f_j\}_{\mathcal{P}_j \in \mathcal{P}}$ in the simulation of the real world above. Equivalently, the honest parties receives all items $\{f_j\}_{\mathcal{P}_j \in \mathcal{P}}$ in the ideal execution. Thus, resulting that the outputs of a successful protocol in the both worlds are indistinguishable. Since the simulator produces the same output as the adversary, on behalf of the malicious parties, their outputs are always indistinguishable in both worlds.

Case 2. The protocol is aborted. If the simulator $\mathcal{S}$ does not have enough TTPs without $\perp$ in their databases after **Resolve 3**, then it sends $ABORT$ to U_{fs}^{ϕ}. Thus, S acting as the ideal adversary, malicious parties do not receive the items in the ideal world. Since the adversary $\mathcal{A}$ in the real world only have the random encrypted items, it does not acquire the items as well. Similarly, the honest parties do not get the items in both ideal and real world where the protocol is aborted. Consequently, the outputs of an aborted protocol in the both worlds are indistinguishable. □

This concludes the proof of the theorem. □

Privacy against TTP: We inherit this property from the previous study [Alper and Küpçü, 2021] as well. If parties use secure and authenticated channels to send their encrypted items VE, then TTPs never have them and cannot learn the items themselves.

Chapter 7

INSTANTIATION

DMFE relies on verifiable and deniable (n, n)-threshold encryption scheme, which we used one based on ElGamal encryption, another verifiable encryption scheme with ($PkGen$, $PkEnc$,$PkDec$), and a PVSS algorithm, which we used Scrape from [Cascudo and David, 2017] with a modification. Scrape [Cascudo and David, 2017] uses a weaker secrecy definition, however, we need a stronger secrecy definition (IND2-Secrecy) for our protocol. Thus, we apply Generic Transformation from [Heidarvand and Villar, 2009] to achieve this stronger secrecy property. In this section, we show how we used the algorithms in an instantiation of our protocol.

We use a group with two independent generators, $\mathbb{G} =< g >=< g_2 >$ of a prime order p where it is hard to compute discrete logarithms, and a security parameter l, are used.

7.1 Setup Phase

TTPs will run the key generation algorithm $PkGen$ to create $(sk_{\mathcal{TP}_i}, pk_{\mathcal{TP}_i})$ and publish their public keys. The TTPs will be ordered.

The parties will run $ThGen(l, n, n)$ algorithm to create their secrets and joint public key. Each party $\mathcal{P}_i$ will choose $m_{\mathcal{P}_i} \in \mathbb{Z}_p$ and sends $h_i = g^{m_{\mathcal{P}_i}}$ to all other parties. Then, the joint public key $pk = \Pi g^{m_{\mathcal{P}_i}}$ and the joint verification key $\nu = \{g^{m_{\mathcal{P}_1}}, g^{m_{\mathcal{P}_2}}, ..., g^{m_{\mathcal{P}_n}}\}$ are decided.

7.2 Encrypted Item Exchange Phase

In this phase, each party $\mathcal{P}_i$ creates verifiable encryption of their item f_i as follows:

- $\mathcal{P}_i$ runs $ThEnc(pk, f_i) \longrightarrow VE = (a, b)$ where $a = g^r$ and $b = f_i \Pi g^{m_{\mathcal{P}_i} r}$ for a random $r \in \mathbb{Z}_p$.

- Then $\mathcal{P}_i$ creates the proof of the encryption using $ProveEnc$.

After creating VE and $VEProof$, $\mathcal{P}_i$ sends these to receiver party.

Each receiver party $\mathcal{P}_j$ checks $VEProof$ and if it fails, then $\mathcal{P}_j$ aborts and does not continue with the next phase.

The verification for the item uses the following relation:

$$\mathbb{R}_{item} = \{(f,(F,c))|F(f,c) = VALID\}$$

The verification function can vary depending on the type of the exchanged item.

7.3 Decryption Share Exchange Phase

In this phase, each party has the VEs and verified them. Then each $\mathcal{P}_i$ creates the decryption shares and distributed shares as follows:

- $\mathcal{P}_i$ runs $ThDShare(m_{\mathcal{P}_i}, pk, VE_{j\to u}) \longrightarrow d^i_{j\to u}$, where $d^i_{j\to u} = a^{m_{\mathcal{P}_i}} = g^{m_{\mathcal{P}_i} r}$, where $d^i_{j\to u}$ is the decryption share to create decryption key for $b = f_i \Pi g^{m_{\mathcal{P}_i} r}$.

- Then $\mathcal{P}_i$ computes the distributed decryption shares and their proofs to send them before decryption shares. First $\mathcal{P}_i$ picks $K \in \mathbb{Z}_p$ and picks a $t-1$ degree polynomials $F_i \in \mathbb{Z}_p[x]$ such that $F(x) = a_0 + a_1.x + ... + a_{t-1}.x^{t-1}$, $F(0) = a_0 = K$. Then $\mathcal{P}_i$ computes the followings:

  - Interim Secrets: $s_u = F(u)$ for $u = 1, ..., m$
  - Encrypted secrets: $\hat{s}_u = pk_k^{s_u}$ where pk_k is the public key of $\mathcal{TP}_k$
  - Commitments: $A_u = g_2^{s_u}$
  - Proofs: $(A_1, .., A_m, e, z_1, ..., z_m)$ where e is the single challenge for $1 \leq u \leq m$ and $z_u = w_u - s_u e$ where $w_u \leftarrow \mathbb{Z}_p$. e is computed from random oracle $H(A_1, pk_1, ..., A_m, pk_m, a_{1,1}, a_{2,1}, ..., a_{1,m}, a_{2,m})$ where $a_{1,u} = g_2^{w_u}$ and $a_{2_u} = pk_k^{w_u}$ using the Fiat-Shamir heuristic [Amos Fiat, 1988]. These values are used to show the following relation:
    $R_{dleq} = \{(s^u, (\mathbb{G}, p, g_2, A_u, pk_k, \hat{s}_u | s^u = \log_{g_2} A_u = \log_{pk_k} \hat{s}_u\}$.

– Transformation: $T = g^K g^{m_{\mathcal{P}_i} r} = g^{K+m_{\mathcal{P}_i} r}$ where r is the random number which used to create VE.

$\mathcal{P}_i$ sends encrypted secrets and the proofs to $\mathcal{P}_j$ for verification. $\mathcal{P}_j$ runs verification algorithm to verify as follows:

– Generates $a_{1,u} = g_2^{z_u} A_u^e$ and $a_{2,u} = pk_u^{z_u} \hat{s}_u^e$

– Checks $H(A_1, \hat{s}_1, ..., A_m, \hat{s}_m, a_{1,1}, a_{2,1}, ..., a_{1,m}, a_{2,m}) \stackrel{?}{=} e$ where $H(.)$ is the random oracle used to create e.

– From the dual code $C^\perp$ of given (t,m)-Threshold Shamir Secret Sharing, generates a random codeword $c^\perp = (c_1^\perp, ..., c_m^\perp)$.

– Checks $\prod_{i=1}^{m} A_i^{c_i^\perp} \stackrel{?}{=} 1$.

Return *valid* if all checks pass, else return *invalid*.

If $\mathcal{P}_j$ does not receive the items before t_1 or cannot verify them, it *ABORTS*. It complains to TTPs by executing **Resolve 1**. Otherwise, it continues to send the decryption share.

- $\mathcal{P}_i$ runs $ThDSProve(m_{\mathcal{P}_i}, pk, VE_{j\to u}, d^i_{j\to u}) \longrightarrow p^i_{j\to u} = (W_{1,1}, W_{1,2}, c, s)$. First $\mathcal{P}_i$ sends $(W_{1,1}, W_{1,2}) = (g^{r'}, a^{r'})$ for randomly chosen $r' \in \mathbb{Z}_p$ by $\mathcal{P}_i$. Then $\mathcal{P}_j$ sends a challenge $c \in \mathbb{Z}_p$ for verification and $\mathcal{P}_i$ calculates $s = r' - m_{\mathcal{P}_i} c$.

- The receiver $\mathcal{P}_j$ runs $ThDsVerify(v_i, VE, d^i_{j\to u}, p^i_{j\to u})$ which outputs $VALID$ if

$$W_{1,1} = g^{r'} = g^s pk_i^c = g^{r'-m_{\mathcal{P}_i} c} g^{m_{\mathcal{P}_i} c} = g^{r'}$$
$$W_{1,2} = a^{r'} = a^s {d^i_{j\to u}}^c = a^{r'-m_{\mathcal{P}_i} c} a^{m_{\mathcal{P}_i} c} = a^{r'}$$

This is the interactive version of the verification [Chaum and Pedersen, 1992].

- For the protocol, we use the non-interactive version of the verification [Blum

et al., 2019].

$$\begin{aligned} ThDSProve(m_{\mathcal{P}_i}, VE_{j\to u}, d^i_{j\to u}) \longrightarrow p^i_{j\to u} &= (c, D) \\ &= (H(W_{1,2}, W_{1,2}), r' - m_{\mathcal{P}_i} c \bmod p) \\ &= (H(g^{r'}, a^{r'}), r' - m_{\mathcal{P}_i} c \bmod p) \end{aligned}$$

for a randomly chosen $r' \in \mathbb{Z}_p$ and $ThDSVerify(v_i, VE, d^i_{j\to u}, p^i_{j\to u})$ outputs $VALID$ only if $c = H(g^D h_i^c, a^D {d^i_{j\to u}}^c)$.

- Both of them shows the relation $R_{dleq} = \{(m_{\mathcal{P}_i}, (\mathbb{G}, p, g, v_i, a, d_i | m_{\mathcal{P}_i} = \log_g v_i = \log_a d_i\}$.

- If $\mathcal{P}_i$ gets all decryption shares from parties without any conflict, it decrypts VE as follows:

$$\frac{b}{\Pi g^{m_{\mathcal{P}_i} r}} = \frac{x h^r}{g^{r \Sigma m_{\mathcal{P}_i}}} = \frac{x h^r}{h^r} = x$$

- If there is a conflict and $\mathcal{P}_i$ gets the distributed decryption shares from TTPs, it decrypts as above after reconstruction of the decryption shares as follows:

$$d = \Pi_{i \in S} s_i^{l_{S,i}(0)}$$

where $l_{S,i}(X)$ is a Lagrange polynomial equal to 0 at $\omega_j \in S$ for $i \neq j$, and 1 at ω_i.

7.4 Reconstruction

If there is a conflict and parties need to reconstruct the decryption share from distributed secret shares, the parties will need to collect at least t shares from TTPs and computes the followings:

- Each TTP $\mathcal{TP}_k$ sends the decrypted shares $\tilde{s_u} = \hat{s}^{\frac{1}{sk_k}} = g^{s_u}$ and $Proof_{i,u} = DLEQ(g, pk_k, \tilde{s}_u, \hat{s}_u)$ to $\mathcal{P}_i$.

- If $\mathcal{P}_i$ verifies the proof and collects at least t shares, then it reconstructs the decryption key $g^{m_{\mathcal{P}_j} r}$ by Lagrange Interpolation as follows:

$$\prod_{\mathcal{TP}_k} (\tilde{s}_u)^{\lambda_u} = \prod_{\mathcal{TP}_k} g^{F(u)\lambda_u} = g^{F(0)} = g^K$$

$$g^{m_{\mathcal{P}_i}} = T.(g^K)^{-1} = \frac{T}{g^K}$$

where $\lambda_u = \prod_{j \neq u} = \frac{j}{j-u}$ are the Lagrange coefficients.

Chapter 8

PERFORMANCE ANALYSIS

We do not provide the code of the full protocol, however, we use the performance of the primitives (see Table 8.1) used in the protocol to compare our performance with previous work [Alper and Küpçü, 2021]. For primitives, we use ZKSK(v0.0.3) [Lueks et al., 2019], Charm [Akinyele et al., 2013] libraries and Scrape [Hanquez, 2017] as in the original paper [Cascudo and David, 2017]. The previous protocol [Alper and Küpçü, 2021] uses Cashlib [Sarah Meiklejohn, 2010] instead of ZKSK [Lueks et al., 2019]. Performance benchmarks are done on Apple Macbook Pro laptop with 2.6GHz i5 processor and 8GB of RAM running MacOS Big Sur 11.7.10 and the average of 3 runs is taken. We use 1024-bits security parameters for public-key encryption scheme, the default in the ElGamal of Charm [Akinyele et al., 2013].

8.1 Computation Performance of Primitives

We provide the performance numbers for El-Gamal [ElGamal, 1985] based verifiable encryption from Charm [Akinyele et al., 2013] library and Zero-Knowledge proofs of knowledge from ZKSK [Lueks et al., 2019] library. For PVSS algorithm, we use Haskell [Hanquez, 2017] implementation.

Computation	**Time (ms)**
Item encryption with h using El Gamal (Enc/Dec)	1.70/0.60
VE(1 item) using El Gamal (Prove/Verify)	1.36/0.90
VD(10 items) using SCRAPE (Escrow/Validating)	5.47/5.37
VD(10 items) using SCRAPE (Decrypting/Verifying-decrypted)	3.57/2.03
VD(6 or more items) using SCRAPE (Reconstruct)	0.72

Table 8.1: Performance numbers of the protocol primitives.

8.2 Computation Overhead

The numbers are calculated from the performance numbers of primitives. Table8.2 and Table 8.3 show that the total computation time per party for complete and ring topologies, when there is no conflict. The # of TTPs is fixed to 10 in Table 8.2 and # of parties is fixed to 10 in Table 8.3. In the Complete Topology, the bottleneck is creating the decryption shares (and dealing them) and it depends on the number of parties.

#	**Complete Topology (ms)**	**Ring Topology (ms)**
2	15.7	15.7
3	39.08	26.54
4	73.20	37.38
5	118.06	48.22
6	173.66	59.06
7	240.00	69.90
8	317.08	80.74
9	404.90	91.58
10	503.46	102.42

Table 8.2: Calculated computation overhead with 10 TTPs.

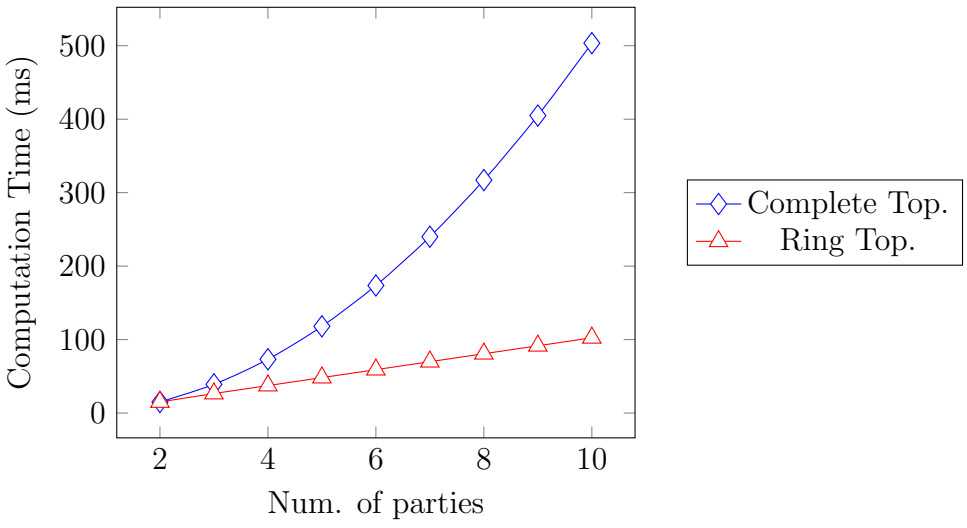

Figure 8.1: Calculated computation overhead with 10 TTPs.

#	**Complete Topology (ms)**	**Ring Topology (ms)**
2	143.46	33.54
3	167.43	36.63
4	215.16	45.72
5	253.68	54.48
6	291.57	61.41
7	326.49	67.53
8	407.34	83.34
9	466.02	92.82
10	503.46	102.42

Table 8.3: Calculated computation overheads with 10 parties.

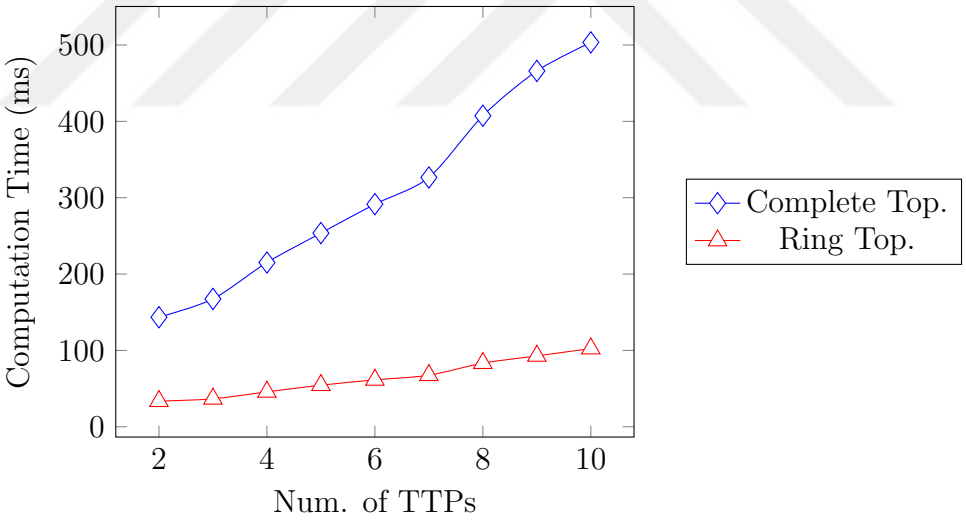

Figure 8.2: Calculated computation overhead with 10 parties.

The resolution steps are not included of the calculation given in Table 8.2 and Table 8.3. Table 8.4 shows the performance of the protocol when the honest parties need to reconstruct decryption keys when different number of parties does not send the decryption key at the last step.

#	**Complete Topology (ms)**	**Ring Topology (ms)**
1	563.30	148.58
2	615.59	200.87
3	667.88	253.16
4	720.17	305.45
5	772.46	357.74
6	824.75	410.03
7	877.04	462.32
8	929.33	514.61
9	981.62	566.90
10	1033.91	619.19

Table 8.4: Calculated computation overheads with 10 parties and 10 TTPs, where # is the number of parties who does not send decryption key.

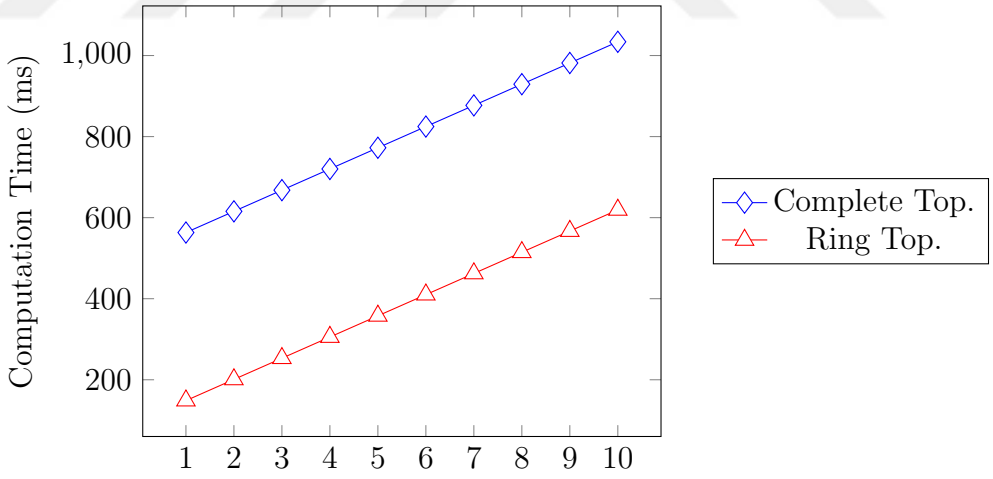

Figure 8.3: Calculated computation overheads with 10 parties and 10 TTPs, where # is the number of parties who does not send decryption key.

Figure 8.3 shows the increase caused by reconstruction of decryption shares per party when different # of parties does not send their decryption shares. The increase is linear since reconstructing only adds $O(n)$ computation cost.

8.3 Comparison with the previous work

Since there is no other solution for multi-party multi-TTP fair exchange with PVSS (without blockchain), we are not able to compare the complexities or performance numbers. For multi-party solutions, the lower bound is $O(n^2)$ using single TTP [Alper and Küpçü, 2021]. There are 2 changes which increases the computation and communication complexity in our protocol compared to [Alper and Küpçü, 2021].

Theoretical View

The computational complexity of distribution, verification and reconstruction are $4m$, $5m$ and $5t+3$, respectively, when using Scrape [Cascudo and David, 2017], where m is the number of TTPs and t is threshold. Therefore, using m TTPs multiplies the computational complexity of step 3 by $O(m)$. Also the communication with TTPs is multiplied by $O(m)$ since the parties interact with m TTPs instead of single TTP. The reconstruction of decryption share causes aincrease by $O(m)$ as well. Thus, the total complexity is $O(n^2) \times O(m) + O(m)$, i.e. $O(n^2m)$.

Numerical View

As we discussed the previous section, dealing the secret and verifying it need more operations and time. Therefore, it is expected to be slower. Figure 8.4 shows the difference between previous work [Alper and Küpçü, 2021] and proposed solution, with fixed number of TTPs. The numbers are directly taken from the paper [Alper and Küpçü, 2021], where they used a DELL Latitude E7240 laptop with a 2.10GHz i7 processor and 8GB of RAM running Ubuntu 16.04 LTS. We used Apple Macbook Pro laptop with 2.6GHz i5 processor and 8GB of RAM running MacOS Big Sur 11.7.10. In both experiments, 1024-bit security parameter is taken for public-key encryption scheme.

Though the results for ring topology do not change much, the number of TTPs affects the performance of the complete topology since we need to deal each decryption shares for each verifiable encryption. The complexity with single TTP is $O(n^2)$ and the number of TTPs changes to $O(n^2m)$, which can be seen in Figure 8.4.

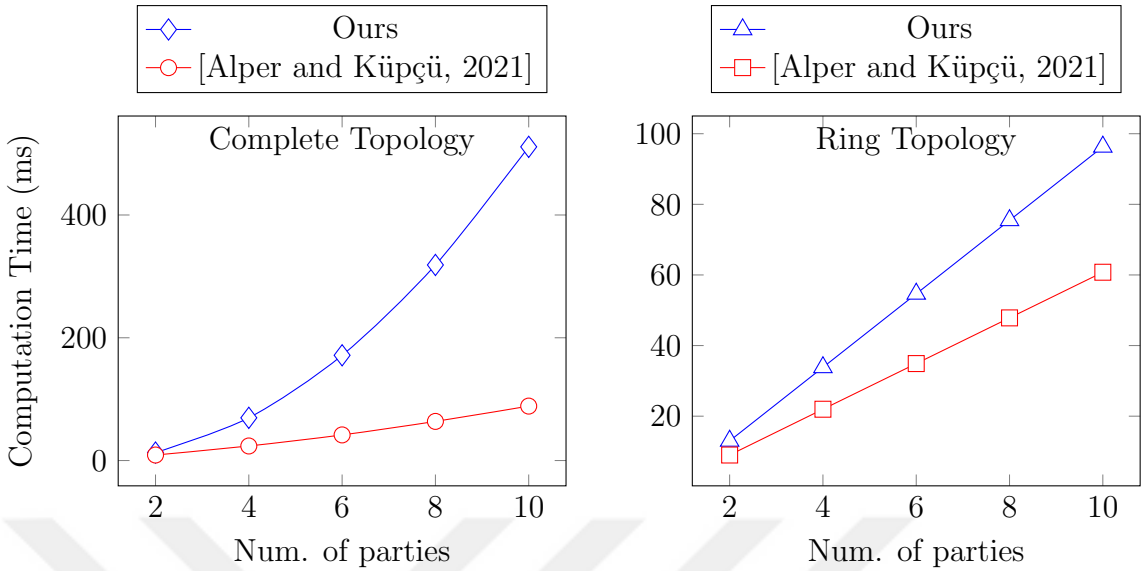

Figure 8.4: Computation Time comparison with the previous work [Alper and Küpçü, 2021] with complete topology and ring topology. Our DMFE uses 10 TTPS.

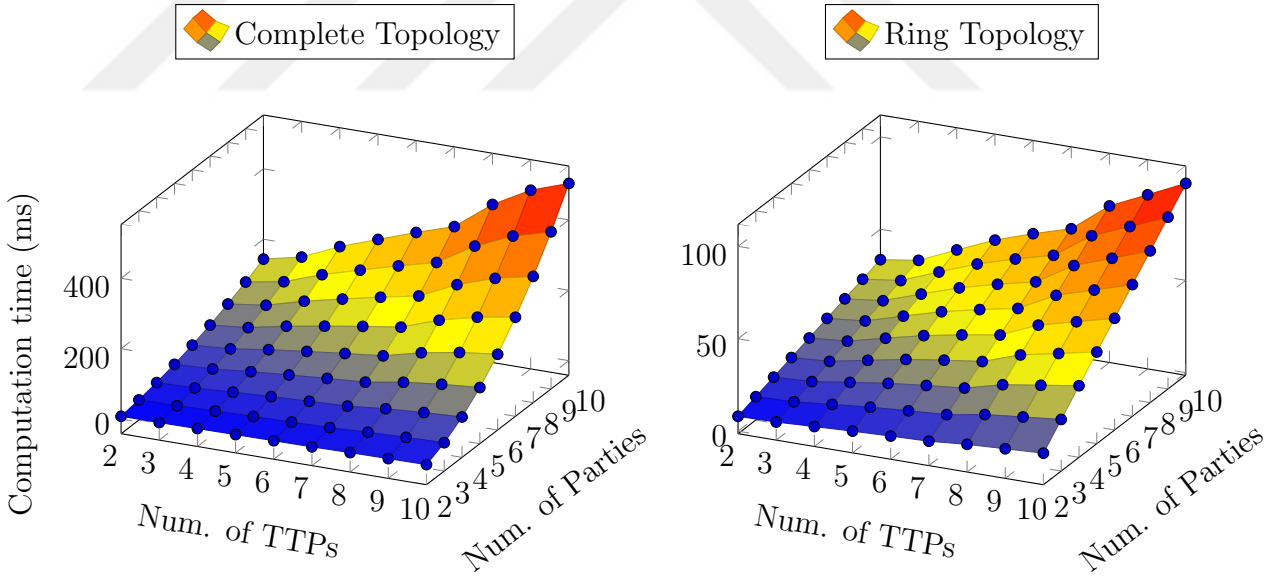

Figure 8.5: Computation Time with 2-10 parties and 2-10 TTPs.

Figure 8.5 shows the comparision different number of TTPs and parties. The number of parties becomes the bottleneck for the protocol.

Chapter 9

CONCLUSION

In this thesis, we proposed Distributed Multi-Party Fair Exchange Protocol as a novel solution for decentralization of the trusted third party in optimistic multi-party setting. We presented the detailed explanation of the protocol and instantiation with the algorithms. We also provided the fairness proof and discussed the performance analysis comparing with previous work. Our protocol is the first instance of optimistic multi-party multi-TTP fair exchange.

We extent the previous study [Alper and Küpçü, 2021] by inheriting the structure and implementation and build our solution on top of it. We inherit the following contributions of [Alper and Küpçü, 2021]:

- Our TTPs do not learn anything about the exchanged files and are used only for fairness.
- They intervene only in case of conflict, so the protocol is optimistic.
- We do not require *broadcasting*.
- Our protocol can be used in any topology. Our contribution does not effect and is not effected by the change of topology.
- Our protocol provides fairness against honest parties even with $n-1$ malicious parties, where n is the number of parties.

Our contribution is providing this with also $t-1$ malicious TTPs, where t is the threshold of PVSS scheme. Our additional cost is only for communication with the TTPs and creating the shares for them, which is expected.

In our protocol, we distribute the responsibility of optimistic TTP of [Alper and Küpçü, 2021] to optimistic multiple TTPs using publicly verifiable secret sharing

[Cascudo and David, 2017]. This requires additional cost of communicating with multiple TTPs. With a single TTP, we create a single decryption key and shares with the TTP. With multiple TTPs, we deal the decryption share to m pieces and shares with m TTPs. This additional cost is inevitable, thus our solution becomes the state of art.

BIBLIOGRAPHY

[Akinyele et al., 2013] Akinyele, J. A., Garman, C., Miers, I., Pagano, M. W., Rushanan, M., Green, M., and Rubin, A. D. (2013). Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3:111–128.

[Alper and Küpçü, 2021] Alper, H. K. and Küpçü, A. (2021). Optimally efficient multi-party fair exchange and fair secure multi-party computation. *ACM Transactions on Privacy and Security*, 25(1):1–34.

[Amos Fiat, 1988] Amos Fiat, A. S. (1988). How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, volume 86, pages 186–194. Springer.

[Asokan et al., 1997] Asokan, N., Schunter, M., and Waidner, M. (1997). Optimistic protocols for fair exchange. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 7–17.

[Avizheh et al., 2022] Avizheh, S., Haffey, P., and Safavi-Naini, R. (2022). Privacy-preserving fairswap: Fairness and privacy interplay. *Proceedings on Privacy Enhancing Technologies*, 1:417–439.

[Avoine and Vaudenay, 2004] Avoine, G. and Vaudenay, S. (2004). Optimistic fair exchange based on publicly verifiable secret sharing. In *Australasian Conference on Information Security and Privacy*, pages 74–85. Springer.

[Bao et al., 1998] Bao, F., Deng, R., and Mao, W. (1998). Efficient and practical fair exchange protocols with off-line ttp. In *Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No.98CB36186)*, pages 77–85.

[Bao et al., 1999] Bao, F., Deng, R., Nguyen, K. Q., and Varadharajan, V. (1999). Multi-party fair exchange with an off-line trusted neutral party. In *Proceedings. Tenth International Workshop on Database and Expert Systems Applications. DEXA 99*, pages 858–862. IEEE.

[Ben-David et al., 2008] Ben-David, A., Nisan, N., and Pinkas, B. (2008). Fairplaymp: a system for secure multi-party computation. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 257–266.

[Blum et al., 2019] Blum, M., Feldman, P., and Micali, S. (2019). Non-interactive zero-knowledge and its applications. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 329–349.

[Camenisch and Damgård, 2000] Camenisch, J. and Damgård, I. (2000). Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In *Advances in Cryptology—ASIACRYPT 2000: 6th International Conference on the Theory and Application of Cryptology and Information Security Kyoto, Japan, December 3–7, 2000 Proceedings 6*, pages 331–345. Springer.

[Camenisch and Damgård, 1998] Camenisch, J. and Damgård, I. B. (1998). Verifiable encryption and applications to group signatures and signature sharing. *BRICS Report Series*, 5(32).

[Camenisch and Shoup, 2003] Camenisch, J. and Shoup, V. (2003). Practical verifiable encryption and decryption of discrete logarithms. In *Advances in Cryptology-CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings 23*, pages 126–144. Springer.

[Canetti, 2000] Canetti, R. (2000). Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13:143–202.

[Cascudo and David, 2017] Cascudo, I. and David, B. (2017). Scrape: Scalable randomness attested by public entities. In *Applied Cryptography and Network Security: 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings 15*, pages 537–556. Springer.

[Chaum and Pedersen, 1992] Chaum, D. and Pedersen, T. P. (1992). Wallet databases with observers. In *Annual international cryptology conference*, pages 89–105. Springer.

[Dziembowski et al., 2018] Dziembowski, S., Eckey, L., and Faust, S. (2018). Fairswap: How to fairly exchange digital goods. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 967–984.

[Eckey et al., 2020] Eckey, L., Faust, S., and Schlosser, B. (2020). Optiswap: Fast optimistic fair exchange. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 543–557.

[ElGamal, 1985] ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472.

[Feldman, 1987] Feldman, P. (1987). A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438. IEEE.

[Ferrer-Gomila and Hinarejos, 2021] Ferrer-Gomila, J.-L. and Hinarejos, M. F. (2021). A multi-party contract signing solution based on blockchain. *Electronics*, 10(12):1457.

[Franklin and Reiter, 1997] Franklin, M. K. and Reiter, M. K. (1997). Fair exchange with a semi-trusted third party. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 1–5.

[Gao et al., 2019] Gao, H., Ma, Z., Luo, S., and Wang, Z. (2019). Bfr-mpc: a blockchain-based fair and robust multi-party computation scheme. *IEEE access*, 7:110439–110450.

[González-Deleito and Markowitch, 2001] González-Deleito, N. and Markowitch, O. (2001). An optimistic multi-party fair exchange protocol with reduced trust requirements. In *International Conference on Information Security and Cryptology*, pages 258–267. Springer.

[Guo et al., 2019] Guo, L., Li, X., and Gao, J. (2019). Multi-party fair exchange protocol with smart contract on bitcoin. *Int. J. Netw. Secur.*, 21(1):71–82.

[Hanquez, 2017] Hanquez, V. (2016-2017). Pvss-haskell. https://github.com/input-output-hk/pvss-haskell.

[Heidarvand and Villar, 2009] Heidarvand, S. and Villar, J. L. (2009). Public verifiability from pairings in secret sharing schemes. In *Selected Areas in Cryptography: 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers 15*, pages 294–308. Springer.

[Katz et al., 2013] Katz, J., Maurer, U., Tackmann, B., and Zikas, V. (2013). Universally composable synchronous computation. In *Theory of Cryptography Conference*, pages 477–498. Springer.

[Khill et al., 2001] Khill, I., Kim, J., Han, I., and Ryou, J. (2001). Multi-party fair exchange protocol using ring architecture model. *Computers & Security*, 20(5):422–439.

[Kılınç Alper and Küpçü, 2021] Kılınç Alper, H. and Küpçü, A. (2021). Coin-based multi-party fair exchange. In *International Conference on Applied Cryptography and Network Security*, pages 130–160. Springer.

[Küpçü, 2013] Küpçü, A. (2013). Distributing trusted third parties. *ACM SIGACT News*, 44(2):92–112.

[Küpçü and Lysyanskaya, 2010] Küpçü, A. and Lysyanskaya, A. (2010). Optimistic fair exchange with multiple arbiters. In *European Symposium on Research in Computer Security*, pages 488–507. Springer.

[Lueks et al., 2019] Lueks, W., Kulynych, B., Fasquelle, J., Bail-Collet, S. L., and Troncoso, C. (2019). zksk: A library for composable zero-knowledge proofs. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society (WPES@CCS)*, pages 50–54.

[Mut-Puigserver et al., 2020] Mut-Puigserver, M., Cabot-Nadal, M. A., and Payeras-Capellà, M. M. (2020). Removing the trusted third party in a confidential multiparty registered edelivery protocol using blockchain. *IEEE Access*, 8:106855–106871.

[Pagnia et al., 1999] Pagnia, H., G¨artner, F. C., et al. (1999). On the impossibility of fair exchange without a trusted third party. Technical report, Citeseer.

[Park et al., 2003] Park, J. M., Chong, E. K., and Siegel, H. J. (2003). Constructing fair-exchange protocols for e-commerce via distributed computation of rsa signatures. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 172–181.

[Payeras-Capella et al., 2019] Payeras-Capella, M. M., Mut-Puigserver, M., and Cabot-Nadal, M. A. (2019). Blockchain-based system for multiparty electronic registered delivery services. *IEEE Access*, 7:95825–95843.

[Pedersen, 1991] Pedersen, T. P. (1991). A threshold cryptosystem without a trusted party. In *Advances in Cryptology—EUROCRYPT'91: Workshop on the Theory and Application of Cryptographic Techniques Brighton, UK, April 8–11, 1991 Proceedings 10*, pages 522–526. Springer.

[Sandıkkaya et al., 2016] Sandıkkaya, M. T., Ovatman, T., and Harmancı, A. E. (2016). Design and formal verification of a cloud compliant secure logging mechanism. *IET Information Security*, 10(4):203–214.

[Sarah Meiklejohn, 2010] Sarah Meiklejohn, C. Chris Erway, A. K. T. H. A. L. (2010). Zkpdl: A language-based system for efficient zero-knowledge proofs and electronic cash. In *19th USENIX Security Symposium (USENIX Security 10)*.

[Schoenmakers, 1999] Schoenmakers, B. (1999). A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings*, pages 148–164. Springer.

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

[Shoup and Gennaro, 2002] Shoup, V. and Gennaro, R. (2002). Securing threshold cryptosystems against chosen ciphertext attack. *Journal of Cryptology*, 15(2):75–96.

[Srivatsa et al., 2005] Srivatsa, M., Xiong, L., and Liu, L. (2005). Exchangeguard: A distributed protocol for electronic fair-exchange. In *19th IEEE International Parallel and Distributed Processing Symposium*, pages 10–pp. IEEE.

[Stadler, 2001] Stadler, M. (2001). Publicly verifiable secret sharing. In *Advances in Cryptology—EUROCRYPT'96: International Conference on the Theory and Application of Cryptographic Techniques Saragossa, Spain, May 12–16, 1996 Proceedings*, pages 190–199. Springer.

[Wu and Tseng, 2011] Wu, T.-Y. and Tseng, Y.-M. (2011). A pairing-based publicly verifiable secret sharing scheme. *Journal of systems science and complexity*, 24(1):186–194.

[Zhang et al., 2023] Zhang, L., Kan, H., Qiu, F., and Hao, F. (2023). A publicly verifiable optimistic fair exchange protocol using decentralized cp-abe. *The Computer Journal*, page bxad039.

[Zhao and Qin, 2012] Zhao, Y. and Qin, Z.-g. (2012). An optimistic protocol for distributed fair exchange. In *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 395–399. IEEE.

[Zhong et al., 2019] Zhong, H., Sang, Y., Zhang, Y., and Xi, Z. (2019). Secure multi-party computation on blockchain: An overview. In *International symposium on parallel architectures, algorithms and programming*, pages 452–460. Springer.