MARCOS VINICIUS MACIEL DA SILVA

SECURETRADE: A SECURE PROTOCOL BASED ON TRANSFERABLE E-CASH FOR EXCHANGING CARDS IN P2P TRADING CARD GAMES

Dissertation submitted to Escola Politécnica da Universidade de São Paulo in partial fulfillment of the requirements for the degree of Master of Science.

São Paulo
2016

MARCOS VINICIUS MACIEL DA SILVA

SECURETRADE: A SECURE PROTOCOL BASED ON TRANSFERABLE E-CASH FOR EXCHANGING CARDS IN P2P TRADING CARD GAMES

Dissertation submitted to Escola Politécnica da Universidade de São Paulo in partial fulfillment of the requirements for the degree of Master of Science.

Concentration area:

Computer Engineering

Supervisor:

Marcos Antonio Simplicio Junior

São Paulo
2016

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catalogação-na-publicação

Silva, Marcos Vinicius Maciel da
SecureTrade: A secure protocol based on transferable e-cash for exchanging cards in P2P trading card games / M. V. M. Silva -- versão corr. -- São Paulo, 2016.
143 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Criptologia 2.Algoritmos 3.Segurança de computadores 4.Jogos eletrônicos I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

RESUMO

Jogos de cartas colecionáveis (TCG, do inglês *Trading Card Game*) diferem de jogos de cartas tradicionais principalmente porque as cartas não são compartilhadas em uma partida. Especificamente, os jogadores usam suas próprias cartas (obtidas, e.g., por meio de compra ou troca com outros jogadores), as quais correspondem a um subconjunto de todas as cartas criadas pelo produtor do jogo. Embora a maioria dos TCGs digitais atuais dependam de um terceiro confiável (TTP, do ingês *Trusted Third-Party*) para prevenir trapaças durante trocas, permitir que os jogadores troquem cartas de maneira segura sem tal entidade, como é o caso em um cenário *peer-to-peer* (P2P), ainda é uma tarefa desafiadora. Possíveis soluções para esse desafio podem ser baseadas em protocolos de moeda eletrônica, mas não sem adaptações decorrentes dos requisitos diferentes de cada cenário: por exemplo, TCGs devem permitir que usuários joguem com as suas cartas, não apenas que passem-nas adiante como ocorre com moedas eletrônicas. Neste trabalho, são apresentados e discutidos os principais requisitos de segurança para trocas de cartas TCGs e como eles se relacionam com moedas eletrônicas. Também é proposto um protocolo eficiente que permite trocas de cartas sem a necessidade de um TTP e com suporte a privacidade. A construção usa como base um protocolo seguro de moeda eletrônica e um protocolo de assinatura-P adaptado para utilizar emparelhamentos assimétricos, mais seguros que os simétricos. De acordo com os experimentos realizados, o protocolo proposto é bastante eficiente para uso na prática: são necessários apenas 5 MB para armazenar um baralho inteiro, enquanto a preparação do mesmo leva apenas alguns segundos; a verificação das cartas, por sua vez, é mais rápida que a duraçãao comum de uma partida e pode ser executada em plano de fundo, durante a própria partida.

ABSTRACT

Trading card games (TCG) are distinct from traditional card games mainly because, in the former, the cards are not shared among players in a match. Instead, users play with the cards they own (e.g., that have been purchased or traded with other players), which correspond to a subset of all cards produced by the game provider. Even though most computer-based TCGs rely on a trusted third-party (TTP) for preventing cheating during trades, allowing them to securely do so in the absence of such entity, as in a Peer-to-Peer (P2P) scenario, remains a challenging task. Potential solutions for this challenge can be based on e-cash protocols, but not without adaptations, as those scenarios display different requirements: for example, TCGs should allow users to play with the cards under their possession, not only to be able to pass those cards over as with digital coins. In this work, we present and discuss the security requirements for allowing cards to be traded in TCGs and how they relate to e-cash. We then propose a concrete and efficient TTP-free protocol for trading cards in a privacy-preserving manner. The construction is based on a secure transferable e-cash protocol and on a P-signature scheme converted to the asymmetric pairing setting. According to our experimental results, the proposed protocol is quite efficient for use in practice: an entire deck is stored in less than 5 MB, while it takes a few seconds to be prepared for a match; the verification of the cards, on its turn, takes less time than an usual match, and can be performed in background while the game is played.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

CID	Class identifier
CL	Camenisch-Lysyanskaya
CRS	Common reference string
CSAT	Circuit satisfiability
DDH	Decisional Diffie-Hellman
DDHI	Decisional Diffie-Hellman inversion
DH	Diffie-Hellman
DHE	Diffie-Hellman exponent
DSA	Digital signature algorithm
ECA	Elliptic curve point addition
ECC	Elliptic curve cryptography
ECDSA	Elliptic curve digital signature algorithm
ECM	Elliptic curve point multiplication
FA	Full anonymity
FFA	Finite field addition
FFM	Finite field multiplication
FlexDHE	Flexible Diffie-Hellman exponent
FFC	Finite field cryptography
FXE	Extended finite field exponentiation

FXM	Extended finite field multiplication
GS	Groth-Sahai
HSDH	Hidden strong Diffie-Hellman
HT	Hyper Threading
IFC	Integer factorization cryptography
P2P	Peer-to-peer
PA	Perfect anonymity
PC	Pairing computation
PPE	Pairing product equation
RSA	Rivest-Shamir-Adleman
SA	Strong anonymity
SNARK	Succinct non-interactive arguments of knowledge
SXDH	Symmetric external Diffie-Hellman
TCG	Trading card game
TTP	Trusted third party
UID	Unique identifier
VRF	Verifiable random function
WA	Weak anonymity

LIST OF SYMBOLS

A	P-signature public key
$\mathcal{A}$	Adversary
$\mathfrak{A}$	Game auditor
$\mathfrak{B}$	Bank
C	Groth-Sahai commitment (in $\mathbb{G}_1$)
$\mathfrak{C}$	Registration center
$\mathfrak{card}$	Card
$\mathfrak{coin}$	Coin
CID	Card class identifier
$\mathcal{CS}$	List of deposited coins
D	Groth-Sahai commitment (in $\mathbb{G}_2$)
$\mathcal{DS}$	List of double-spenders
E	Groth-Sahai output map
EC	Elliptic curve
e	Bilinear pairing
eq	Equation
$\mathbb{F}$	Field
$\mathcal{F}$	Evaluation algorithm
G	Generator of $\mathbb{G}_1$

$\mathbb{G}$	Group
$\mathcal{G}$	Setup algorithm
$\mathfrak{G}$	Game server
gk	Groth-Sahai setup information
$gsparams$	Groth-Sahai parameters
H	Generator of $\mathbb{G}_2$
$\mathcal{H}$	Cryptographic hash function
h	Number of hops
I	Group identity element
$\mathfrak{info}$	Public information from the transference
K	P-signature commitment
k	Security level
L	Wallet size
l	Index of the coin from the wallet
M	RSA Modulus
$\mathfrak{M}$	Card market
m	Message
N	Number of equations
$\mathbb{N}$	Set of natural numbers
n	Number of messages to sign
O	Big-oh (algorithm complexity notation)
$\mathcal{O}$	Point at infinity

$open$	Opening
$owner$	Card owner
$\mathcal{P}$	Prover algorithm
$\mathfrak{P}$	Player
$pparams$	P-signature parameters
p	Field characteristic
pk	Public key
q	Sample size of instances (Sec. 3.5)
q	Group order
R	Matrix of openings to commitments in $\mathbb{G}_1$ (Sec. 3.6 only)
R	Summary of transference information
r	Ownership tag (Sec. 3.9 and forward)
$\mathcal{RS}$	List of reported cards
S	Matrix of openings to commitments in $\mathbb{G}_2$ (Sec. 3.6 only)
S	Coin serial number
s	Serial seed (Sec. 3.9 and forward)
sk	Private key
T	Transference tag
t	Transference tag seed
U	Groth-Sahai common reference string component in $\mathbb{G}_1$ (Sec. 3.6)
U	P-signature public key component
U_0	P-signature public key component

U_1	P-signature public key component
$\mathfrak{U}$	User
UID	Card unique identifier
V	Groth-Sahai common reference string component in $\mathbb{G}_2$ (Sec. 3.6)
V	Card validity information
$\mathcal{V}$	Verifier algorithm
W	P-signature witness
$\mathfrak{wallet}$	Wallet
$\mathbb{Z}_q$	Integers modulo q
α	P-signature private key component
β	P-signature private key component
Δ	Matrix of integer exponents
δ	Pairing product equation exponent constant
γ	P-signature private key component
ι	Groth-Sahai input map
κ	Groth-Sahai commitment
Λ	Pairing setting
ν	Groth-Sahai common reference string scalar component
Π	List of transferences
π	Groth-Sahai proof of knowledge component in $\mathbb{G}_2$
σ	P-signature
ς	Groth-Sahai common reference string

Υ	Randomization matrix for Groth-Sahai proofs
υ	Groth-Sahai common reference string scalar component
θ	Transformation
Θ	Set of allowable transformations
ϕ	Proof of knowledge
φ	Groth-Sahai proof of knowledge component in $\mathbb{G}_1$
ψ	Group homomorphism
Ω	P-signature public key component
ω	P-signature private key component

CONTENTS

1 INTRODUCTION

A trading card game (TCG) is a type of card game in which, instead of using a fixed deck, each player creates his/her own deck from a subset of all cards made available by the game provider (SIMPLICIO JR. et al., 2014). During a match, players usually do not share their cards with their opponents; hence, as many different cards may exist, one important part of the game is to build decks that support a target strategy or game style. To build better decks, users may either trade cards with other users or purchase them directly from the game provider. To improve their revenue, in the last years some providers have expanded their markets beyond the realm of physical cards, creating digital versions of their games. This is the case, for example, of "Magic: the Gathering™" [1], one of the first TCGs ever released. There are also TCGs that do not even have a physical counterpart, but only a digital version, like the case of "Hearthstone: Heroes of Warcraft" [2]. An extensive list of TCGs can be found in ONLINE GAMES KINGDOM.

1.1 Motivation

The video game industry is very lucrative and is spreading even more. An annual report about sales and usage data in video games in the U.S. (ESA, 2015) presents that, while computer games are stagnating, the revenue of social and casual games is increasing. More people are playing multiplayer games, and for longer time, specially

[1] http://magic.wizards.com/en/content/magic-duels
[2] http://us.battle.net/hearthstone/en/

in mobile devices. The complete success multiplayer games depend on how the players are treated, since they usually present a different business model: instead of profiting from sales, the profit may be in-game purchases or periodical fees (PRITCHARD, 2000; YAN; CHOI, 2002). The more success these games achieve, higher attempts to hack and cheat will come. And in this case, cheating undermines success.

To set matches and avoid cheating, digital TCGs typically use a client-server architecture, where the centralized system acts as card market and also as referee for the matches between players. When considering mobile applications, however, a peer-to-peer (P2P) architecture may present advantages over the client-server one (PITTMAN; GAUTHIERDICKEY, 2013; SIMPLICIO JR. et al., 2014). The reason is that a client-server model obliges players to have a continuous Internet connection when trading or playing, preventing them to do any of those actions otherwise. Using a local connection also unload the central area of the network, concentrating at the peripheral area (more idle). It helps saving energy and avoids denial of service due to overload in central hubs. Besides, the game provider does not need to keep an updated list of each player's current cards (unless it so desires); after all, trading can happen without connection to the server, so the players themselves are responsible for validating the ownership of cards. If the game protocols are designed so they do not depend on a trusted third party (TTP) to prevent cheating, on the other hand, then a local connection would be enough, bringing convenience to users and also to game providers.

1.2 Goals

The main goals of this work is to define the requirements for secure card exchange in P2P-based TCGs and then provide a solution that addresses those requirements in an efficient manner. Due to similarities to transferable e-cash, such requirements and a concrete protocol instantiation can be adapted from e-cash schemes, at least to fulfill the most basic properties of trading cards.

1.3 Related works

Playing traditional card games in a P2P model was originally proposed in the context of *mental poker* (SHAMIR; RIVEST; ADLEMAN, 1981), and different solutions were proposed since then (for a survey, see (ROCA; FEIXAS; DOMINGO-FERRER, 2006)). These works also served as basis for TTP-free solutions for TCGs, such as Match+Guardian (PITTMAN; GAUTHIERDICKEY, 2013) and SecureTCG (SIMPLICIO JR. et al., 2014), which allow the detection of cheating attempts during a match with two or more players. Despite those advances concerning *in-game* cheating, such protocols were constructed so that they still depend on a trusted entity for each card trading event, leaving the task of reducing this dependence as a subject for future work.

Arguably, the closest solutions to this problem are the ones described in (GAUTHIERDICKEY; RITZDORF, 2014; GAUTHIERDICKEY; RITZDORF, 2012), in which fair transference in multiplayer games are allowed. Both schemes rely on the assumption that all game items (cards, in TCGs) from each player are known to other players. When crossing the information of transferences and the public inventory of the sender, it is possible to audit if any player has illegally duplicated some item. If applied to TCGs, this would violate the confidentiality of the player's cards, violating their privacy because each card could be easily linkable to its owner. This is also particularly undesirable in TCGs because the confidentiality of the users' decks is commonly part of their strategy: after all, this prevents players from setting up a deck specifically designed to counter their opponents' decks. Since this public inventory is critical to the correctness of the protocol, an adaptation of these existent protocols will not achieve the confidentiality required in a TCG protocol.

1.4 Contribution

Aiming to tackle the above issues, in this work we: (1) define the requirements for secure card trading in TCGs in a P2P architecture; and (2) instantiate a protocol that fulfills those requirements, allowing players to detect cheating attempts when exchanging cards with each other even before a match starts.

Trading cards in a TTP-free manner is a problem that resembles the issue tackled by transferable e-cash protocols (CHAUM; PEDERSEN, 1993a; FUCHSBAUER; POINTCHEVAL; VERGNAUD, 2009; CAMENISCH; HOHENBERGER; LYSYANSKAYA, 2005; BALDIMTSI et al., 2015), where the cards replace the digital money. For example, as in e-cash, a player should be able to anonymously trade cards with other players without the need of a TTP mediating the transactions. However, if a player sends the same card to two or more peers (i.e., "double-spends" it), this should be detectable and the transgressor's anonymity should be revoked. Nevertheless, TCGs also have additional requirements, as there is no concept similar to "playing with owned cards" in the context of e-cash. To the best of our knowledge, the specialized literature has no clear list of security requirements that apply to card trading, which so far has hindered further progress in this area.

We propose a scheme based on existing transferable e-cash protocols (namely, (CAMENISCH; HOHENBERGER; LYSYANSKAYA, 2005) and (BELENKIY et al., 2009)), with the required adaptations for allowing players to: (1) purchase cards from the game provider in a privacy-preserving manner, meaning that a card cannot be linked to any user unless its owner generates a proof of ownership; (2) use the cards they own in a match; (3) trade cards with other players; (4) verify the validity of a card presented by any player without the intervention of a TTP, independently of the number of previous owners the card has ever had; (5) let the game provider know about cheating events, such as a user playing with a card that has already been handed over to another

user. Since the resulting protocol is transparent to how the matches themselves are handled, it can also be integrated with in-game cheating-detection mechanisms such as the aforementioned Match+Guardian or SecureTCG, thus allowing the construction of a secure P2P-based TCG environment.

In addition to the main contributions mentioned, some improvements to the underlying protocol were by-products from our main goals. Namely, the P-signature scheme (a special digital signature scheme, presented in Section 3.8) was originally proposed in a symmetric pairing setting. However, this type of pairing suffers from attacks that reduces the security level of protocols constructed over it (BARBULESCU et al., 2014). For this reason, we convert the original scheme to one based on an asymmetric pairing setting of the same security level, using the method proposed by (ABE et al., 2014). Besides better security, this results in a more efficient protocols to prove knowledge on the commitments and on the signature; more precisely, the resulting protocols are more concise and faster due to a reduced number of equation and pairing computations.

1.5 Outline

The rest of this document is organized as follows. Chapter 2 presents the characteristics of TCGs, the main subject of our work; it also compares TCGs with e-cash, which allows the system requirements for a P2P TCG to be defined. Chapter 3 defines the notation used, the building blocks for our system and the security assumptions of a concrete protocol instantiation. The proposed scheme based on e-cash is then defined in Chapter 4, and its security and efficiency are analyzed in Chapter 5. Finally, we present our concluding remarks and the ideas for future work in Chapter 6.

2 TRADING CARD GAMES

To define the requirements for a secure card exchange protocol, it is necessary to discuss the terminology associated to this environment, as well as the desired functionalities of a P2P TCG. In this chapter, we discuss the TCG environment, which is the focus of the proposed solution, and how it works in the digital world. We begin with details regarding which features are expected from a TCG, defining the associated terminology. Then, we analyze the architectures proposed in the literature for in-game cheating-detection and how the entities of the system are thereby represented, since, for better compatibility of the discussion, the same entities are employed in this work. Finally, we compare the game system with that of a transferable e-cash, enlightening similarities and distinctions between them, which allows the requirements of a secure P2P TCG to be defined.

2.1 TCGs: Scenario description

Each TCG has its own set of rules, which vary from the order of steps in a match to how the cards are used. The cards also vary from game to game, since each may have different properties and some cards are produced in limited amount. Some games also allow players to use several cards of the same kind in their deck, depending on specific rules for those matches. Some common game styles are the following (WIZARDS, 2014):

- *Constructed deck*: Players build their decks with cards of their own, thus relying

on the cards they had before the match.

- *Uniform deck*: Players receive the same set of cards from the game provider and choose which ones are going to be put in their decks.
- *Sealed deck*: Players receive random sets of cards from the game provider and select the cards to be put in their decks.
- *Draft deck*: Players receive a single set o cards and, in a round-robin fashion, each one takes a turn to pick a card from the set, passing the remaining cards to the next player in the queue.

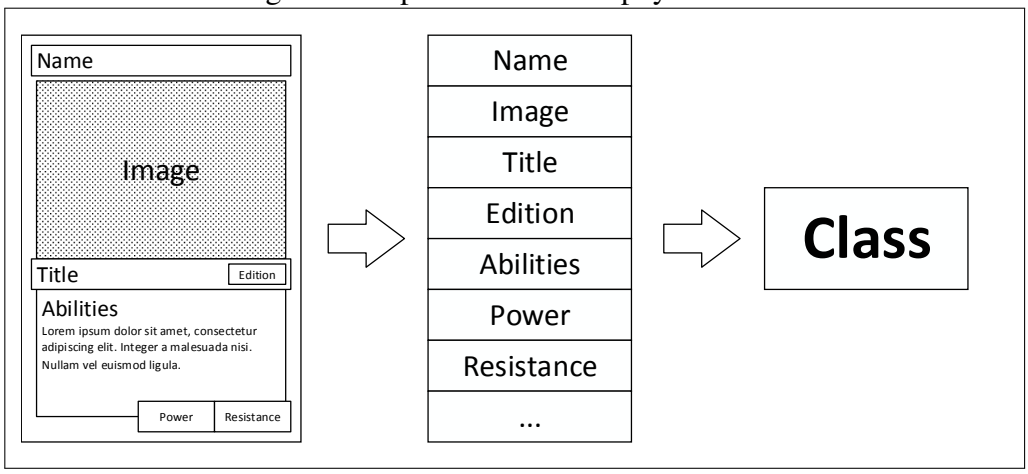
Despite those differences, the method to obtain cards is basically the same in all TCGs: cards can be bought directly from the game provider or can be traded with other users.

2.1.1 Card representation

Cards (card) have different effects in the game, and each card may have a distinct description of its properties (e.g., name, attack/defense power, abilities, or reactions to certain game events). If they present the same set of properties, they belong to the same *class* of cards (as shown in Figure 1), similarly to coins having the same value in e-cash schemes. Cards of the same class have the same class identifier number (CID), and in games that allow players to have several cards of the same class, each *instance* of these cards has a unique identifier (UID), that univocally identifies that card in the system. This UID can then be used by in-game cheat-detection protocols to identify the card (PITTMAN; GAUTHIERDICKEY, 2013; SIMPLICIO JR. et al., 2014).

Each card should belong to only one player, and some information (*owner*) allows the card's owner to be identified. This information may be mutable, since, after each trade, ownership is passed to the receiver.

Figure 1: Representation of a physical card



Source: Author

Finally, each card depends on some validity information V, which can be verified without the direct intervention from a trusted party. This information is used to verify that the card has been correctly stamped by the market, as well as to detect if some player has illegally used a duplicate of some card after having handed it over in a trade.

Combining this information, a card in our work can be represented by the tuple (also represented in Figure 2):

$$\mathfrak{card} = (UID, CID, owner, V)$$

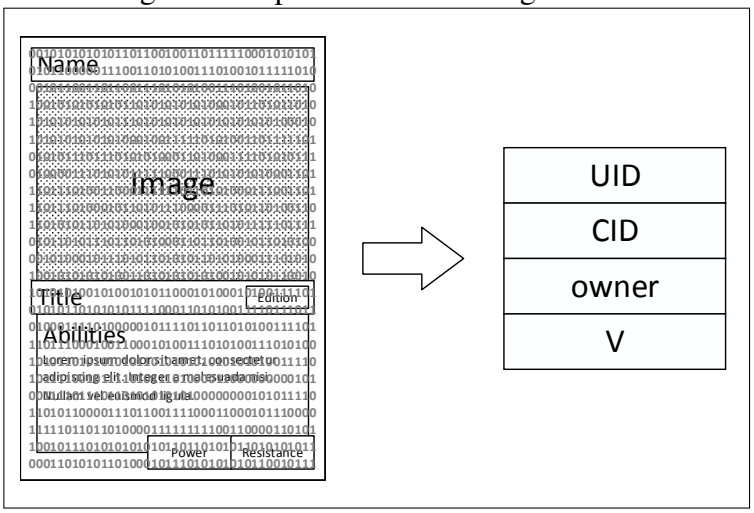
The manner by which this information is specified in our construction is described in Chapter 4, in which a concrete instantiation of the proposed protocol is described.

2.1.2 Architecture

Following the notation of Pittman and GauthierDickey (2013) and Simplicio Jr. et al. (2014), the architecture (presented in Figure 3) of a P2P TCG encompasses a game server and the players.

The game server ($\mathfrak{G}$) is responsible for any action that requires a trusted authority

Figure 2: Representation of a digital card



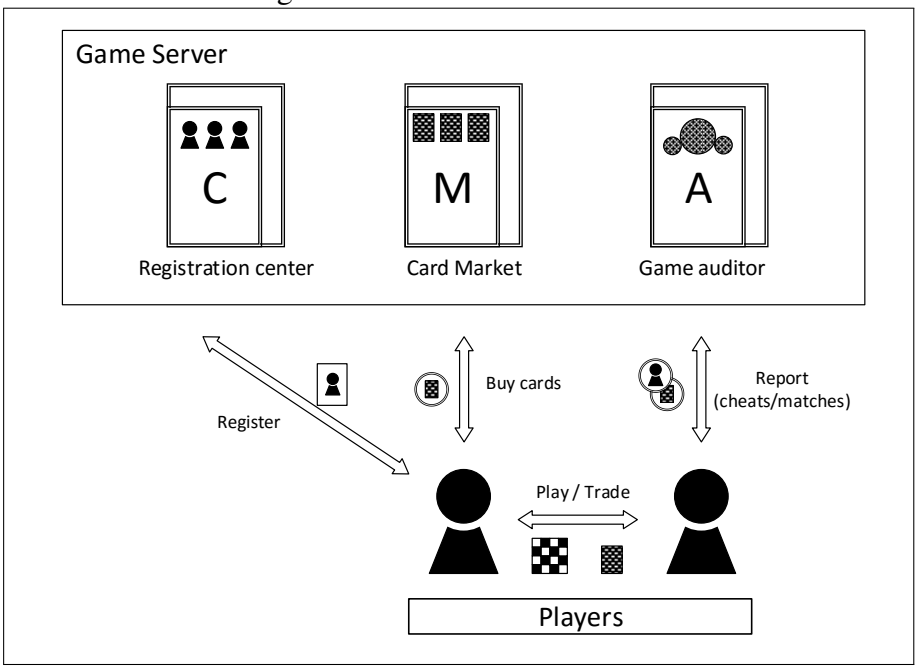
Source: Author

or centralized information storage. One of its primary roles is to serve as a *registration center* ($\mathfrak{C}$) for players: to enroll in the system, a user must register with a unique identifier (e.g., an e-mail or social security number) and provide his/her public key; the game server then generates a digital certificate to assert this information, allowing anyone to verify who are the system's authorized users.

The game server also acts as a *card market* ($\mathfrak{M}$), being responsible for selling and digitally signing cards, so the buyer can prove that a card is valid, as well as its ownership. As a result, the server does not need to keep record of the cards possessed by each player, as ownership varies with time and, as proposed in this work, trading may occur without the server's knowledge. The server is also responsible for informing players of the valid *CID*s of all cards available, as new releases usually add several new cards to the game.

Finally, the server also plays the role of *game auditor* ($\mathfrak{A}$), verifying claims regarding cheating attempts and eventually punishing those responsible for misbehavior. For example, in (PITTMAN; GAUTHIERDICKEY, 2013; SIMPLICIO JR. et al., 2014), the players may send after-match information to the server to prove that a user tried to cheat, e.g., by modifying the sequence or contents of their deck during a match.

Figure 3: P2P TCG Architecture



Source: Author

If a player sends to the server the list of cards employed by an adversary, the server should also be able to verify the usage of cards that were not under a malicious player's possession at the time of the match (e.g., because he/she had traded it earlier). Providing such after-match data is actually very common, as this information is normally required to rank players depending on the number of victories in matches.

Any other action that does not require a trusted third party (TTP), such as playing the game or trading cards, ideally should be allowed to be performed in a purely P2P fashion, while still being protected by cheating-detection mechanisms. As in-game cheating is quite thoroughly covered in (SIMPLICIO JR. et al., 2014), in this work we focus only on cheating-detection during card trading.

2.2 Comparison with e-cash

The security issues that appear when trading cards are somewhat similar to those faced by transferable e-cash. Indeed, both systems must provide some sort of *balance*,

Table 1: Parallel methods of E-cash and P2P TCG

E-cash	**P2P TCG**
Setup	Setup
Register	Register
Mint	Stamp
Spend	Trade
Self-spend *	Play
Deposit	Refresh/Report *
Identify	Identify

* Methods not originally developed for e-cash

Source: Author

so that the number of elements (coins or cards) of the system should not grow without the central server's authorization. Hence, no user should be able to produce more elements than what the central server has emitted, which could be done by forging a new element or duplicating an existing valid one. As shown in Table 1, many actions supported by card trading and transferable e-cash protocols are also similar: stamping new cards is similar to minting new coins, while trading cards is equivalent to spending coins. The anonymity of the users that participated in the trades of a card is also desirable in both scenarios for protecting the privacy of the users in those transactions.

It is, thus, reasonable to build a secure card trading protocol from a transferable e-cash scheme. In this case, like coins, the card's portion that indicates ownership (*owner*), grows in size with each transference (CHAUM; PEDERSEN, 1993a), or need to be stored somewhere else to prevent such growth (e.g., in a receipt (FUCHSBAUER; POINTCHEVAL; VERGNAUD, 2009)). To avoid indefinite growth, players may *refresh* their cards, which is equivalent to deposit a coin and get a new, mint version of it. TTP-free transferability also raises the problem of duplicating existing elements, an issue that cannot be prevented but can be detected so that the culprit is identifiable when the coin is deposited at the central server. More precisely, in case of double-spending in transferable e-cash schemes, the central server is able to revoke the anonymity of the user responsible for misbehavior, and only of that user, independently of how many owners the coin had before or after it was copied.

In the context of TCGs, however, the double spending problem is a more complicated issue because players may not only trade, but also use a card (i.e., play with it during matches) without transferring its ownership. Therefore, TCGs also need mechanisms for detecting a scenario in which a user irregularly plays with a card that has been previously traded. As further discussed in Section 4, this can be accomplished if the server crosses the information about refreshed cards with those received from match reports. Hence, refreshing cards benefits both honest players and the game server: the former get a shorter copy of the card, which is less computationally expensive to verify and trade, while the latter is able to audit trades by using the information stored in the cards submitted for refreshing. The same mutual benefit applies to the match reports: honest players who win matches can raise their ranks by informing their victories to the server; honest players who lose matches can make sure the opponent played fairly; and the server can audit if some refreshed or traded card has been illicitly used in a match. It should, thus, be quite easy to encourage players to provide such information often to the server.

There are, thus, five main types of cheating that can appear when cards are traded in TCGs:

1. *Double-refresh*: refreshing the same card twice, obtaining several valid instances of the same card but purchasing a single one;
2. *Double-trade*: sending copies of the same card to different users;
3. *Trade-then-play*: playing with a card that has already been passed to another user;
4. *Refresh-then-trade*: refreshing a card $\mathfrak{card}$ to obtain a mint version of it, $\mathfrak{card}'$, but then trading copies of $\mathfrak{card}$ with other users; and
5. *Refresh-then-play*: refreshing a card $\mathfrak{card}$ to obtain a mint version of it, $\mathfrak{card}'$, but then using $\mathfrak{card}$ in matches with other players.

Whenever a player provides a match's report information or refresh their cards, it is trivial to the game server to identify that someone has cheated, even if the cheater is in collusion with other players. For example, suppose that a player $\mathfrak{P}_D$ duplicates a card and sends it to two other players $\mathfrak{P}_{R,0}$ and $\mathfrak{P}_{R,1}$. Even if both know that this card is duplicated, when each one plays with the duplicated card against honest players $\mathfrak{P}_{H,0}$ and $\mathfrak{P}_{H,1}$, the "self-spend" operation will modify the card, creating two different instances of it. When the honest players report the usage of these cards after a match is over, the game server is able to identify $\mathfrak{P}_D$ as the one responsible for the *Double-trade*, without revealing any information about the other owners. Other collusion situations are handled similarly, as they are basically a direct result of the underlying e-cash protocol's resistance against collusion.

2.3 System requirements

From the previous discussion, we can postulate that the following security and usability requirements must be met in by secure P2P-based TCG system. They are extensions to e-cash properties, as defined in (CAMENISCH; HOHENBERGER; LYSYANSKAYA, 2005), extended to fulfill the TCG environment.

- *Verifiable stamping*: The card market must stamp cards, so their validity and ownership can be verified without the need of contacting the central server.
- *TTP-free transferability*: Players should be able to trade cards with each other without the intervention of a TTP, and the new ownership can also be verified without the need of contacting a trusted server.
- *Anonymity*: Suppose that user $\mathfrak{U}_0$ purchases a given card $\mathfrak{card}$, and then that card is repeatedly traded among a set of users $\{\mathfrak{U}_j\}_{j=1\ldots n}$ before the last owner, $\mathfrak{U}_{n+1}$, informs the server about this ownership. In this case, the server only learns the identity of $\mathfrak{U}_{n+1}$, while $\mathfrak{card}$'s previous owners remain anonymous. In addition,

during this process user $\mathfrak{U}_j$ only learns the identity of $\mathfrak{U}_{j-1}$ and $\mathfrak{U}_{j+1}$, i.e., each player only knows the users with which he/she has traded directly. In collusion with others users, the server cannot prove that the card has belonged to any other player.

- *Balance*: The number of cards in the system cannot grow unless the central server stamps new cards, with invalid duplicates being detected and removed.

- *Cheat detection*: Players cannot trade a card more than once without losing their anonymity toward the server, nor play with a card after having traded it.

- *Exculpability*: The game server, even if in collusion with users, cannot falsely prove that an honest user has cheated, i.e., the cheating-detection mechanism only allows to identify users who have duplicated a card (either for trading or playing with it).

2.4 Summary

In this Chapter we have presented the scenario of our work, introducing how cards are represented in a secure digital TCG and which roles are necessary in a TCG-oriented architecture, following the notation of the literature in this area (in especial, (PITTMAN; GAUTHIERDICKEY, 2013; SIMPLICIO JR. et al., 2014)). We have also compared P2P TCGs with transferable e-cash, which allowed (1) the identification of which situations could be interpreted as cheating attempts and (2) the definition of the main security and usability requirements in this context. This analysis also indicates that a secure protocol for trading cards in P2P TCGs can indeed be built using transferable e-cash schemes as basis, although some are necessary to cover the dissimilarities in the two scenarios' requirements.

3 BUILDING BLOCKS

This chapter presents the mechanisms necessary for a concrete construction of a secure trading protocol for P2P-based TCGs. Specifically, the proposed scheme is based on the transferable e-cash scheme described in (CAMENISCH; HOHENBERGER; LYSYANSKAYA, 2005) and revisited in (BELENKIY et al., 2009), which relies on asymmetric pairings, witness-indistinguishable non-interactive proofs, verifiable random functions and structure-preserving blind signatures. We start with a discussion of the basic definitions of elliptic curves and bilinear pairings, introducing the notation hereby employed and the underlying assumptions on the hardness of some security problems (Sections 3.1 to 3.5); we note that these definitions are quite standard and, thus, readers with background in pairing-based cryptography may prefer to skip those sections. We then describe each of the cryptographic building blocks employed in the aforementioned e-cash scheme (Sections 3.6 to 3.8) before discussing the scheme itself (Section 3.9).

3.1 Notation

Throughout the document, we employ the following basic notation and definitions.

Given a finite set $\mathcal{S}$, $s \leftarrow \mathcal{S}$ denotes the process of selecting an element s of $\mathcal{S}$. When we write $s \xleftarrow{\$} \mathcal{S}$, the element s is sampled uniformly at random from $\mathcal{S}$.

Given two functions $f, g : \mathbb{N} \rightarrow [0, 1]$, we say $f(k) \approx g(k)$ if $|f(k) - g(k)| = O(k^{-c})$ for all constants c (i.e., the between f and g difference is upper bounded by an

exponentially small constant). We say that $f(k)$ is *negligible* if $f(k) \approx 0$, and $f(k)$ is called *overwhelming* if $f(k) \approx 1$.

Given an algorithm $\mathcal{F}$, the execution of $\mathcal{F}$ with input x and output y is written as $\mathcal{F}(x) \rightarrow y$. If $\mathcal{F}$ is an interactive algorithm between parties $\mathcal{A}$ and $\mathcal{B}$, $\mathcal{F}(\mathcal{A}(a) \leftrightarrow \mathcal{B}(b)) \rightarrow y$ is the execution of $\mathcal{F}$ with inputs a for $\mathcal{A}$ and b for $\mathcal{B}$ and output y.

We also consider a cryptographic hash function $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^{2k}$ (e.g., SHA-3 (NIST, 2015)). If $\mathcal{H}$ has more than one input, we assume the inputs are concatenated in the order they are presented.

3.2 Mathematical concepts

For the sake of completeness, and aiming to provide a better understanding of the cryptographic concepts that allow the construction of more complex systems based in elliptic curves and bilinear pairings, in this section we list the basic definitions related to the arithmetic in finite fields. The definitions were extracted from Shoup (2008, ch. 1).

Definition 1 (Abelian group)**.** *An abelian group* $(\mathbb{G}, \star)$ *is a set* $\mathbb{G}$ *together with an operation* $\star$ *on* $\mathbb{G}$ *such that:*

1. $\star$ *is associative, i.e.,* $\forall A, B, C \in \mathbb{G}, A \star (B \star C) = (A \star B) \star C$*;*
2. $\star$ *has a unique identity element I, such that* $\forall B \in \mathbb{G}, A \star I = A = I \star A$*;*
3. $\forall A \in \mathbb{G}$, $\star$ *has an inverse element* A'*, such that* $A \star A' = I = A' \star A$*;*
4. $\star$ *is commutative, i.e.,* $\forall A, B \in \mathbb{G}, A \star B = B \star A$.

This $\star$ operation may be replaced by some named operation in the group, called *addition* (denoted by +) to an additive group (with identity $I = 0$ and inverse $A' = -A$)

or *multiplication* (denoted by $\cdot$) to a multiplicative group (with identity $I = 1$ and inverse $A' = A^{-1}$). The n-fold composition of $\star$ also depends on the notation; in the additive notation, it is represented by $nA = A + \ldots + A$ (n times A) and nA is called *multiple* of A, while in the multiplicative notation it is represented by $A^n = A \cdots A$ (n factors A) and A^n is called a *power* of A. Such notations can be interchangeable without affecting the defined operation.

Definition 2 (Cyclic group)**.** *A multiplicative (resp., additive) group $\mathbb{G}$ is said to be* cyclic *if there is an element $G \in \mathbb{G}$ so that, for any $P \in \mathbb{G}$, there is some integer a satisfying $P = G^a$ (resp., $P = a\,G$). Such element G is called a* generator *of $\mathbb{G}$, and we write $\mathbb{G} = \langle G \rangle$.*

Abelian groups can also be classified depending on their cardinality (the number of their elements).

Definition 3 (Finite and infinite groups)**.** *A group $\mathbb{G}$ is called* finite *(resp.* infinite*) if it contains finitely (resp. infinitely) many elements. The number of elements of a finite group is called its order and is represented by $|\mathbb{G}|$.*

For some element $P \in \mathbb{G}$, the subset $\langle P \rangle$ may not contain all elements of $\mathbb{G}$. If $\langle P \rangle$ is also a group regarding the same operation $\star$ of $\mathbb{G}$, $\langle P \rangle$ is called a subgroup of $\mathbb{G}$.

Definition 4 (Subgroup generated (LIDL; NIEDERREITER, 1997, p. 6))**.** *The subgroup of $\mathbb{G}$ consisting of all powers of the element $G \in \mathbb{G}$ is called the* subgroup generated by *G and is denoted by $\langle G \rangle$. This subgroup is necessarily cyclic. If $\langle G \rangle$ is finite, then its order (represented by $\#G$) is called the* order *of the element G; otherwise it is said that G has* infinite order.

Usually, we work in sets with two operations, addition and multiplication, which lead to the definition of fields, as follows.

Definition 5 (Field)**.** *A field* $(\mathbb{F}, +, \cdot)$ *is a set* $\mathbb{F}$ *together with two operations, addition (denoted by* $+$*) and multiplication (denoted by* $\cdot$*), that satisfy the usual arithmetic properties:*

1. $(\mathbb{F}, +)$ *is an abelian group with additive identity denoted by 0.*
2. $(\mathbb{F}, \cdot)$ *is an abelian group with multiplicative identity denoted by 1.*
3. *The distributive law holds:* $\forall a, b, c \in \mathbb{F}, (a + b) \cdot c = a \cdot c + b \cdot c$.

For some prime n, a field with finite n elements may be represented by $\mathbb{F}_n$, and it is called a *finite field* or a *Galois field* of order n. This prime n (or the prime base p of some power $n = p^b, b > 0$) is the characteristic of $\mathbb{F}_n$ and is defined as:

Definition 6 (Characteristic of finite field)**.** *If* $\mathbb{F}$ *is a field and there is a positive integer* p *such that* $pa = 0$ *for every* $a \in \mathbb{F}$*, then the least such positive integer* p *is called the* characteristic *of* $\mathbb{F}$ *and* $\mathbb{F}$ *is said to have characteristic* p*. If there is no such integer* p*,* $\mathbb{F}$ *is said to have characteristic* 0 *(then* n *is not a prime power and* $\mathbb{F}$ *is not a finite field).*

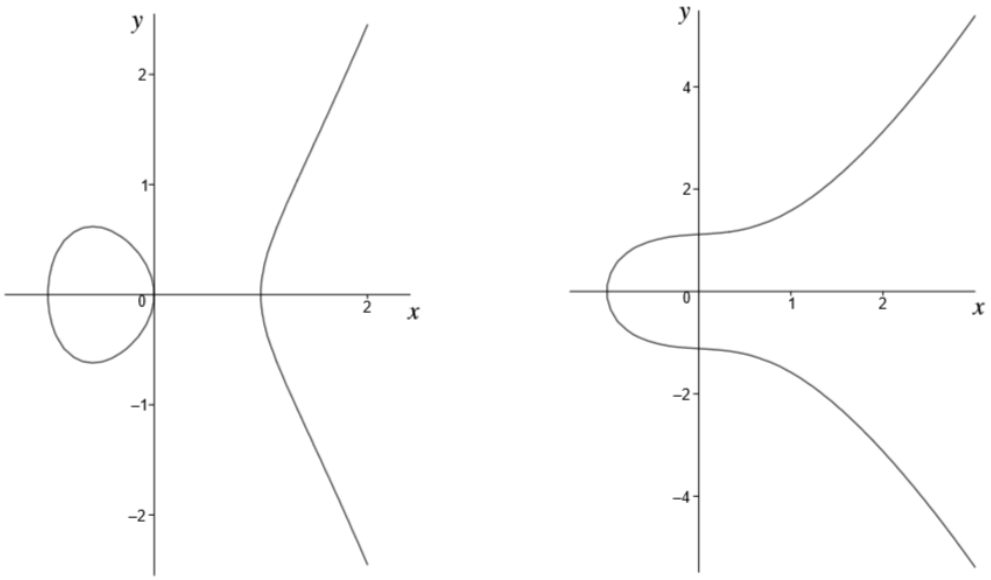
3.3 Elliptic curves

An elliptic curve EC, viewed as a plane curve, is represented by the solution to some cubic equation, as illustrated in Figure 4. Their usage in the context of cryptography was originally proposed for cryptanalysis to factor large integers (LENSTRA JR, 1987) and only later in the construction of cryptographic schemes. In modern cryptography, the usage of elliptic curves involves the selection of a subset of rational points from EC in a way that they have an abelian group operation, which can be more formally described as follows:

Definition 7 (Elliptic curves)**.** *An* elliptic curve *EC defined over a field* $\mathbb{F}_n$ *is the set* $(x, y) \in {\mathbb{F}_n}^2$ *of solutions to the cubic equation:*

$$EC/\mathbb{F}_n : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \quad a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}_n \tag{3.1}$$

Figure 4: Elliptic curves in the real plane



(a) $E_1 : y^2 = x^3 - x$ (b) $E_2 : y^2 = x^3 + \frac{1}{4}x + \frac{5}{4}$

Source: (HANKERSON; MENEZES; VANSTONE, 2004, p. 77)

, together with a "point at infinity" O. Equation 3.1 is in the generalized Weierstrass form, and if the field has prime characteristic larger than 2*, the curve equation can be represented by the Weierstrass equation, presented in Equation 3.2.*

$$EC/\mathbb{F}_n : y^2 = x^3 + ax + b, \quad a, b \in \mathbb{F}_n \tag{3.2}$$

If the cubic equation that represents the curve EC is nonsingular, meaning that it does not have repeated roots (HUSEMÖLLER, 2004, Remarks 2.1 and 5.3), the *point addition* operation (defined by the chord-tangent law of composition) of points in a curve is an abelian group operation in the set of the solutions to the curve equation, and the point O is the identity element (WEIL, 1928).

The use of elliptic curves in cryptography began with the independent works of Koblitz (1987) and Miller (1986), in which they present the elliptic curve equivalent to the discrete logarithm problem.

Definition 8 (Elliptic curve discrete logarithm problem (MILLER, 1986))**.** *Given an elliptic curve EC defined over $\mathbb{F}_n$ and two points $P, Q \in EC$, find an integer a such that*

Q = aP, if such a exists.

One of the main reasons to its use is probably that Elliptic Curve Cryptography (ECC) leads to shorter parameters than more traditional cryptographic schemes, as illustrated in Table 2 for different families of algorithms: protocols based on finite-field cryptography (FFC, a.k.a. protocols based on discrete logarithm), such as Digital Signature Algorithm (DSA) (NIST, 2013) and Diffie-Hellman (DH) (DIFFIE; HELLMAN, 1976), whose parameters are measured in terms of the size of the public key $\overline{pk}$ and of the private key $\overline{sk}$; on integer factorization cryptography (IFC), such as the Rivest Shamir Adleman (RSA) (RIVEST; SHAMIR; ADLEMAN, 1978), measured in terms of the size of the modulus M; and based on elliptic curve cryptography (ECC), such as the Elliptic Curve Digital Signature Algorithm (ECDSA) (NIST, 2013), which is measured in terms of the size of the order $\#G$ of the base point G. In Table 2, column 1 represents the number of steps to break the protocol (for k bits of security, it would take 2^k steps to break it). The ranges in column 4 represent the equivalence between the order of elliptic curves and standardized security levels (rounded down).

Table 2: Comparison between the size of parameters and the resulting security strength for different cryptographic approaches. All values are measured in number of bits.

Bits of security	FFC	IFC	ECC
80	$\overline{pk} = 1024$; $\overline{sk} = 160$	$M = 1024$	$\#G = 160 - 223$
112	$\overline{pk} = 2048$; $\overline{sk} = 224$	$M = 2048$	$\#G = 224 - 255$
128	$\overline{pk} = 3072$; $\overline{sk} = 256$	$M = 3072$	$\#G = 256 - 383$
192	$\overline{pk} = 7680$; $\overline{sk} = 384$	$M = 7680$	$\#G = 384 - 511$
256	$\overline{pk} = 15360$; $\overline{sk} = 512$	$M = 15360$	$\#G = 512+$

Source: adapted from (NIST, 2012a)

As several cryptographic schemes had been constructed over the discrete logarithm problem for cyclic abelian groups, and with the equivalence for the elliptic curve counterpart, it has induced research focused on efficient implementation of elliptic curve operations (HANKERSON; MENEZES; VANSTONE, 2004).

3.4 Bilinear pairings

Constructions of elliptic curves are important not only on its own, but also because they are the basis for more complex constructions, such as bilinear pairings. Generically, a bilinear pairing can be defined as follows:

Definition 9 (Bilinear pairing)**.** *Consider the additive groups $\mathbb{G}_1$ and $\mathbb{G}_2$ with identity O, and the multiplicative group $\mathbb{G}_T$ with identity* 1*, all of prime order q. A* bilinear pairing *is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the properties:*

1. Bilinearity*: $\forall (P, Q) \in \mathbb{G}_1 \times \mathbb{G}_2, \forall a, b \in \mathbb{Z}_q : e(aP, bQ) = e(P, Q)^{ab}$.*
2. Non-degeneracy*: $\forall P \in \mathbb{G}_1, \forall Q \in \mathbb{G}_2, P \neq O_{\mathbb{G}_1}, Q \neq O_{\mathbb{G}_2} : e(P, Q) \neq 1_{\mathbb{G}_T}$.*
3. Computability*: $\forall (P, Q) \in \mathbb{G}_1 \times \mathbb{G}_2, e(P, Q)$ is efficiently computable.*

It is important to note that the source groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are presented in additive notation and $\mathbb{G}_T$ in multiplicative notation. It is usually so because $\mathbb{G}_1$ and $\mathbb{G}_2$ are subgroups of the group of points of some elliptic curves EC_1 and EC_2, whose operation is usually called *point addition*. However, the target group $\mathbb{G}_T$ is presented in the multiplicative notation because its operation is different (and more cumbersome) than that from $\mathbb{G}_1$ and $\mathbb{G}_2$.

The pairings can be classified depending on the relation of the source groups. When both groups are the same, i.e., $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$, it is called a *symmetric* pairing. When $\mathbb{G}_1 \neq \mathbb{G}_2$, it is called an *asymmetric* pairing. More specifically:

Definition 10 (Pairing types (GALBRAITH; PATERSON; SMART, 2008))**.** *Given the bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, it is possible to separate pairings in three basic types:*

***Type 1**: $\mathbb{G}_1 = \mathbb{G}_2$.*

***Type 2**: $\mathbb{G}_1 \neq \mathbb{G}_2$ but there is an efficiently computable homomorphism $\psi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$.*

***Type 3**: $\mathbb{G}_1 \neq \mathbb{G}_2$ and there is no efficiently computable homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$.*

Distinguishing between these types is important because the construction of such pairings depends on the source groups, and the existence of the homomorphism may create differences in the efficiency of the methods in such groups. Type 1 pairings are constructed over supersingular elliptic curves with small embedding degree ($d \leq 6$), and the attack presented in (BARBULESCU et al., 2014) for computing the discrete logarithm in these groups makes these constructions impracticable. Type 3 pairings are at least as efficient as Type 2 pairings, and usually outperform them, and there is always an equivalent complexity assumption in Type 3 as hard as the one in Type 2 when the parameters are appropriately chosen (CHATTERJEE; MENEZES, 2011). Therefore in this work we consider only Type 3 pairings, which will be simply called *asymmetric pairings*. More precisely, along the discussion we consider the asymmetric (Type 3) pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, where the source groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are elliptic curves, all groups have the same prime order q, and there is some map functions from $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ to $\{0,1\}^*$, such that any group can be used as input to the hash function $\mathcal{H}$.

3.5 Security assumptions

The security of cryptographic blocks used in our scheme are based in reducing some attack (e.g., forgery, capability of inverting some function) to problems that are considered hard in the adopted settings (summarized in Table 3). In the constructions hereby discussed, we assume the hardness of the following computational problems:

Definition 11 (Decision Diffie-Hellman (DDH) (BONEH, 1998))**.** *Given a non-trivial triple $(aG, bG, P) \in \mathbb{G}_i^3, i = \{1,2\}$, the* decision Diffie-Hellman *problem is to decide*

whether or not $P = ab\,G$.

Definition 12 (Symmetric external Diffie-Hellman (SXDH) (BALLARD et al., 2005))**.** *The* symmetric external Diffie-Hellman *problem is to solve the DDH in* $\mathbb{G}_1$ *or in* $\mathbb{G}_2$.

Definition 13 (***q***-decisional Diffie-Hellman inversion (q-DDHI) (DODIS; YAMPOLSKIY, 2005))**.** *Given a non-trivial q-uple* $(a\,G, a^2\,G, \ldots, a^{q-1}\,G, P) \in \mathbb{G}_i^q, i = \{1, 2\}$*, the* q-decisional Diffie-Hellman inversion *problem is to determine whether or not* $P = \frac{1}{a}\,G$.

Definition 14 (***q***-hidden strong Diffie-Hellman (q-HSDH) (BOYEN; WATERS, 2007))**.** *Given a non-trivial triple* $(G, U, \omega\,G) \in \mathbb{G}_i^3, i = \{1, 2\}$ *and a set of q triples* $(\frac{1}{\omega+c_j}\,G, c_j\,G, c_j\,U) \in \mathbb{G}_i^3$*, with* $\forall\, j \in [1, q] : c_j \xleftarrow{\$} \mathbb{Z}_q^*$*, the* q-hidden strong Diffie-Hellman *problem is to find an additional non-trivial triple* $(\frac{1}{\omega+c}\,G, c\,G, c\,U) \in \mathbb{G}_i^3$ *such that* $\forall\, j \in [1, q] : c \neq c_j$.

Definition 15 (***n***-Diffie-Hellman exponent (n-DHE) (BONEH; GENTRY; WATERS, 2005))**.** *Given* $\forall\, j \in [0, 2n] \setminus (n+1) : G_j = \alpha^j\,G \in \mathbb{G}_i, i \in \{1, 2\}$ *for some* $\alpha \in \mathbb{Z}_q^*$*, the* n-Diffie-Hellman exponent *problem is to compute the missing element* $G_{n+1} = \alpha^{n+1}\,G \in \mathbb{G}_i$.

Definition 16 (Flexible ***n***-Diffie-Hellman exponent (n-FlexDHE) (IZABACHÈNE; LIBERT; VERGNAUD, 2011))**.** *Given* $\forall\, j \in [0, 2n] \setminus (n+1) : G_j = \alpha^j\,G \in \mathbb{G}_i, i \in \{1, 2\}$ *for some* $\alpha \in \mathbb{Z}_q^*$*, the* flexible n-Diffie-Hellman exponent *problem is to compute a triple* $(\mu\,G, \mu\,G_{n+1}, \mu\,G_{2n}) \in \mathbb{G}_i^3$ *such that* $\mu \neq 0$ *and* $G_{n+1} = \alpha^{n+1}\,G$.

Besides these computational problems, when we convert the signature scheme described in (IZABACHÈNE; LIBERT; VERGNAUD, 2011) from a Type 1 pairing setting to a Type 3 pairing following the methods from (ABE et al., 2014), the associated security assumptions are also modified. When some element has to be part of both source groups in the pairing, they have to be duplicated. If these duplicated elements are elements from the assumption instance, the very assumption is altered to its

Table 3: Computational problems assumed hard for the security of each presented cryptographic tool

Assumption	Cryptographic primitive
SXDH	Proof of knowledge (Section 3.6.4)
q-DDHI (in $\mathbb{G}_1$)	Verifiable Random Function (Sections 3.7.1 and 3.7.2)
co-q-HSDH^{+1}	P-signature (Section 3.8.1)
co-Flex-DH^{+3}	P-signature (Section 3.8.1)
co-n-FlexDHE^{+2n}	P-signature (Section 3.8.1)

Source: Author

asymmetric counterpart, namely a *co-* security assumption. From our conversion, the following assumptions are defined:

Definition 17 (co-***q***-hidden strong Diffie-Hellman (co-q-HSDH^{+1}))**.** *Given a non-trivial tuple* $(G, H, U, \omega G) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1 \times \mathbb{G}_1$ *and a set of* q *quadruples* $(\frac{1}{\omega+c_j} G, c_j G, c_j H, c_j U) \in (\mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1)$*, with* $\forall j \in [1, q] : c_j \xleftarrow{\$} \mathbb{Z}_q^*$*, the* co-$q$-hidden strong Diffie-Hellman *problem is to find an additional non-trivial triple* $(\frac{1}{\omega+c} G, c G, c U) \in \mathbb{G}_1^3$ *or* $(\frac{1}{\omega+c} G, c H, c U) \in (\mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1)$ *such that* $\forall j \in [1, q] : c \neq c_j$*.*

Definition 18 (co-flexible Diffie-Hellman (co-Flex-DH^{+3}))**.** *Given a non-trivial tuple* $(G, a G, b G, H, a H, b H) \in (\mathbb{G}_1^3 \times \mathbb{G}_2^3)$*, the* co-flexible Diffie-Hellman *problem is to find either* $(\mu G, \mu a G, \mu ab G) \in \mathbb{G}_1^3$ *or* $(\mu H, \mu a H, \mu ab H) \in \mathbb{G}_2^3$ *such that* $\mu \neq 0$*.*

Definition 19 (co-flexible ***n***-Diffie-Hellman exponent (co-n-FlexDHE^{+2n}))**.** *Given* $\forall j \in [0, 2n] \setminus (n+1) : G_j = \alpha^j G \in \mathbb{G}_1, H_j = \alpha^j H \in \mathbb{G}_2$ *for some* $\alpha \in \mathbb{Z}_q^*$*, the* co-flexible n-Diffie-Hellman exponent *problem is to compute either* $(\mu G, \mu G_{n+1}, \mu G_{2n}) \in \mathbb{G}_1^3$ *or* $(\mu H, \mu H_{n+1}, \mu H_{2n}) \in \mathbb{G}_2^3$ *such that* $\mu \neq 0$*,* $G_{n+1} = \alpha^{n+1} G$ *and* $H_{n+1} = \alpha^{n+1} H$*.*

3.6 Zero-Knowledge Proof of Knowledge

Proofs of knowledge allow a party to prove knowledge of some secret value without revealing it, which is done by showing a witness satisfying some relation that

depends on the secret. Moreover, a zero-knowledge proof of knowledge does not reveal any information about this secret. A zero-knowledge proof system is a *proof of knowledge* if the *completeness*, *soundness* and *zero-knowledge* properties are satisfied (MENEZES; OORSCHOT; VANSTONE, 1996, Chapter 10.4), which are defined as follows.

Definition 20 (Completeness)**.** *A proof system is* complete *if, given an honest prover and an honest verifier, the protocol succeeds with overwhelming probability (i.e., the verifier accepts the prover's claim).*

Definition 21 (Soundness)**.** *A proof system is* sound *if, given an honest verifier and a dishonest prover trying to impersonate the honest prover (the dishonest prover does not know the secret), the protocol succeeds with negligible probability (i.e., the verifier does not accept the prover's false claim).*

Definition 22 (Zero-knowledge)**.** *A proof system is* zero-knowledge *if there is a polynomial-time simulator which can produce, upon input of the assertions to be proven but without interacting with the real prover, transcripts indistinguishable from those resulting from interaction with the real prover.*

3.6.1 Proof of knowledge schemes: a brief review

In the protocol proposed in this work, we use the Groth-Sahai proof of knowledge (GROTH; SAHAI, 2008). This choice was motivated by the fact that their structure conforms with automorphic signature schemes (more details in Section 3.8), it is efficient when proving sentences in a well-defined language (witnesses that satisfy pairing product equations – PPE), and it guarantees all security properties for an e-cash scheme.

We notice, however, that many other schemes were considered before the choice for Groth-Sahai proofs of knowledge was made. In special, some quite efficient

Table 4: Analysis of proof of knowledge protocols

Method	Analysis
Challenge-response	
Interactive methods	Pros: Most efficient (only group operations) Prove a plethora of algebraic relations Cons: Does not provide transferability
Fiat-Shamir heuristic	Pros: Most efficient (group operations and hash function) Prove a plethora of algebraic relations Cons: Proofs are not sound
CRS	
CSAT Proofs	Pros: Very efficient Prove any NP-problem instance Cons: Problem reduction necessary Reduction can be inefficient
Groth-Sahai proofs	Pros: Efficient Prove a plethora of group equations Cons: Necessity of pairing computations

Source: Author

schemes were considered but discarded due to the lack of required security properties in the target scenario as summarized in Table 4. Among them some interactive proof of knowledge systems were considered. They have already been proposed from a large range of algebraic relations (e.g., discrete logarithm of a group element (CHAUM; EVERTSE; GRAAF, 1988), and equality of two discrete logarithms in different bases (CHAUM; PEDERSEN, 1993b)). These proofs are based on a challenge-response method, so each proof is only accepted by one verifier. For transferable elements, however, one cannot expect any interaction with the parties not directly involved in the current transference. Therefore, they would simply not fit the scenario of secure card trading.

Using the Fiat-Shamir heuristic (FIAT; SHAMIR, 1987), i.e., with random oracles

to simulate a random challenge, it would be possible to transform interactive proofs of knowledge into signatures of knowledge (signatures that endorse the interactive proof of knowledge). However, unfortunately it is not possible to create a sound zero-knowledge signature with this approach due to the very simulatability of the protocol, which allows one to create signature forgeries (OKAMOTO, 1993).

One alternative approach for creating sound zero-knowledge proofs is to employ a *common reference string* (CRS) generated by some trusted party (BLUM; FELDMAN; MICALI, 1988), a strategy that is also adopted in this work. It consists on a trusted entity generating some common information (namely, the CRS), alike a public key, that can be used by all other entities in the system to prove a statement. When first received by the users, the CRS can be signed or proved well-formed (using the proof of knowledge) to guarantee its validity. The advantage of this strategy is that it allows the construction of very efficient systems that prove generic constructions while avoiding the mentioned issues with zero-knowledge signatures (GROTH; OSTROVSKY; SAHAI, 2006a; GROTH; OSTROVSKY; SAHAI, 2006b). Nevertheless, many of the existing schemes that adopt a CRS are built to prove the problem of Circuit Satisfiability (CSAT), which also means that, to prove other NP-problems, the instance must firstly be reduced to CSAT previously to the proof. This reduction may be cumbersome, since the statement to be proven is composed of several bits and, thus, the circuit involved in the proof may end up having from hundreds to thousands of gates.

Since in this work we need to prove knowledge of group elements (e.g., signatures, encryption, discrete logarithm), group-dependent proofs are good alternatives to CSAT proofs. Nevertheless, as shown in (GROTH; SAHAI, 2008), non-interactive proofs can still be performed in an efficient manner when the relation to be proved is a set of equations in some defined format and the witnesses are variables that belong to the solutions set. The equations must be a pairing product equation (PPE), a multi-scalar multiplication equation, or a quadratic equation. In this work, we focus on PPE since

several signature schemes were constructed to be proven in a zero-knowledge fashion by means of these equations. More precisely, in the bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ of order q, a PPE is of the form:

$$\prod_{i=0}^{m} e(X_i, B_i) \prod_{j=0}^{n} e(A_j, Y_j) \prod_{i=0}^{m} \prod_{j=0}^{n} e(X_i, Y_j)^{\delta_{ij}} = c$$

where $A_j \in \mathbb{G}_1$, $B_i \in \mathbb{G}_2$, $c \in \mathbb{G}_T$ and $\delta_{ij} \in \mathbb{Z}_q$ are constants, and $X_i \in \mathbb{G}_1$ and $Y_j \in \mathbb{G}_2$ are variables. To create a proof of knowledge, all variables are committed with a random opening, so that these values can be shown without revealing any information, but are still bound by the relation defined by the PPE.

3.6.2 The Groth-Sahai proof system

The Groth-Sahai proof system is a *witness-indistinguishable* proof of knowledge. Besides satisfying the completeness and soundness properties, it also satisfies the *witness-indistinguishability* property.

Definition 23 (Witness-indistinguishability (FEIGE; SHAMIR, 1990))**.** *A proof system is* witness-indistinguishable *if, given an honest prover, the probability of distinguishing two transcripts generated by the same prover with two different witnesses (i.e., two instances with different solutions) is negligible.*

The algorithms employed by the Groth-Sahai proof system are the following:

- $GSSetup(1^k) \rightarrow (gk, \varsigma)$

  Creates a setup information gk and a CRS ς for input on the proving algorithms, for some security parameter k.

- $GSCommit(gk, \varsigma, P\ [, r]) \rightarrow \kappa$

  Creates a commitment κ that completely hides the input value P with some random opening r.

- $GS\,Prove(gk, \varsigma, \{eq_k\}_{k=1...N}, \vec{X}, \vec{Y}\ [, \vec{r}]\ [, \vec{s}]) \rightarrow \phi$

  Generates a proof $\{\pi_k, \varphi_k\}_{k=1...N}$ such that the input values $(\vec{X}, \vec{Y})$ satisfy the relation defined by the set of equations $\{eq_k\}_{k=1...N}$. Random values $(\vec{r}, \vec{s})$ are used to commit inputs $(\vec{X}, \vec{Y})$, randomizing the proof.

- $GS\,Verify(gk, \varsigma, \{eq_k\}_{k=1...N}, \vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1...N}) \rightarrow \{0, 1\}$

  Verifies if a proof $\{\pi_k, \varphi_k\}_{k=1...N}$ was correctly constructed, and that the values committed to $(\vec{C}, \vec{D})$ satisfy the set of equations $\{eq_k\}_{k=1...N}$.

A witness-indistinguishable proof reveals more information than a zero-knowledge proof, since it is not possible to distinguish a valid instance from the simulated one. Hence, additional restrictions must be defined to guarantee zero-knowledge in Groth-Sahai proofs. First we need a trivial equation to hide the group generator. More specifically, we need to prove that $e(P, H) = e(G, H)$, where $P = 1G$ (resp. $e(G, Q) = e(G, H)$, where $Q = 1H$). Then, each proved variable $X_i \in \mathbb{G}_1$ (resp. $Y_j \in \mathbb{G}_2$) must be duplicated to $X'_i = X_i$ (resp. $Y'_j = Y_j$) and hidden by the equality equation $e(X'_i, Q)\ e(X_i, Q)^{-1} = 1$ (resp. $e(P, Y'_j)\ e(P, Y_j)^{-1} = 1$). Furthermore, the simulator generates a simulated CRS with a simulation trapdoor indistinguishable from the real CRS. This allows simulating proofs for statements without knowing the corresponding witnesses.

For a Groth-Sahai proof instantiation under SXDH assumption (presented in Section 3.6.4), we need 4 elements in $\mathbb{G}_1$ and 4 elements in $\mathbb{G}_2$ for each equation, whereas each variable will be committed to 2 elements in their group.

3.6.3 Malleability of Groth-Sahai proofs

Malleability is a property of some encryption protocols that allows someone with access to a valid ciphertext to *maul* it, i.e., to create another valid ciphertext to the same message or to a transformed message. This notion is analogously applied to digital sig-

natures. Although this property is usually undesirable for cryptographic protocols, as it might allow signature to be forged, for example, controlled-malleability provides interesting features in some scenarios. For example, one can apply homomorphic operations over encrypted messages, in such a manner that some operations performed with the ciphertexts will consistently lead to some operations over the corresponding plaintexts after decryption; it is also possible to re-randomize signatures present in a document, so that they are unlinkable to the original ones when that same document is forwarded from user to user.

The Groth-Sahai proof system presents two levels of malleability: one for its commitments and another for its proofs. In order to achieve zero-knowledge composition, Groth-Sahai proofs produce simulatable commitments based on ElGamal encryption (ELGAMAL, 1984). Due to this fact, the commitments share the malleability property with the encryption scheme. Given a valid commitment κ_w from a witness w, any entity is able to produce another valid commitment $\kappa' \leftarrow \theta_\kappa(\kappa_w)$ to another valid witness $w' \leftarrow \theta_w(w)$, for some valid transformation $\theta = (\theta_\kappa, \theta_w)$. Even though this breaks the integrity of the encrypted data, the proofs cannot be forged unless the forger knows the secret values used to hide the committed values. This composition is used to compose the proof of knowledge of a verifiable random function, as we present in Section 3.7.

On top of that, the entire proof of knowledge is malleable. If we have a proof ϕ that the relation $R(\phi, x)$ holds (e.g., that x is a solution to a PPE), a prover is able to produce another valid proof $\phi' \leftarrow \theta_\phi(\phi)$ for a new solution $x' \leftarrow \theta_x(x)$ to the same relation R, and the relation $R(\phi', x')$ also holds. The transformations $\theta = (\theta_\phi, \theta_x)$ are composed by operations over commitments, equations and proofs (to maul ϕ), and by some transformation that generates another solution to R (that depends on the equation to prove).

Definition 24 (Allowable set of transformations (CHASE et al., 2012))**.** *An efficient relation R is closed under an n-ary transformation $\theta = (\theta_a, \theta_b)$ if for any n-tuple*

$\{(a_1, b_1), \ldots, (a_n, b_n)\} \in R^n$, *the pair* $(\theta_a(a_1, \ldots, a_n), \theta_b(b_1, \ldots, b_n)) \in R$. *If R is closed under* θ, *then we say that* θ *is admissible for R. Let* Θ *be a set of transformations; if for every* $\theta \in \Theta$, θ *is admissible for R, then* Θ *is an* allowable set of transformations.

When a non-interactive proof of knowledge accepts those transformations, it is called a malleable proof of knowledge.

Definition 25 (Malleable proof of knowledge (CHASE et al., 2012))**.** *Let* $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ *be a proof of knowledge for some relation R, and* Θ *be an allowable set of transformations for R. The proof system is* malleable *with respect to* Θ *if there is an efficient algorithm* $\mathcal{F}$ *that on input* $(\varsigma, \theta, \{x_i, \phi_i\}_{i \in [1;n]})$, *where* $\theta \in \Theta$ *is an n-ary transformation,* x_i *is the input for a proof,* ϕ_i *is the proof of knowledge of x, and* $\mathcal{V}(\varsigma, x_i, \phi_i) = 1$ *for all* $i \in [1; n]$, *outputs a valid proof* ϕ' *for the statement* $x = \theta_x(\{x_i\})$ *(i.e., a proof* ϕ *such that* $\mathcal{V}(\varsigma, x, \phi) = 1$*).*

Specifically for the Groth-Sahai proofs, the set of transformations contains transformations on either variables or equations. They allow a prover that knows the committed values to create new proofs that complement or modify the original PPE relation. Namely: merge equations, merge variables, exponentiate variables, add constant equation, remove equation, and remove variable. We refer to (CHASE et al., 2012) for a complete description of these transformations.

3.6.4 A Groth-Sahai instantiation

In what follows, we present a concrete instantiation of Groth-Sahai proof under SXDH assumption for proving PPE.

- $GSSetup(1^k) \rightarrow (gk, \varsigma)$

  For the security parameter k, consider the asymmetric bilinear setting $\Lambda = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H, q)$. The CRS can be produced in two different settings, indistinguishable from each other:

Soundness string: Get random $\upsilon_1, \upsilon_2 \xleftarrow{\$} \mathbb{Z}_q^*$, set $U_1 = (G, \upsilon_1 G) \in \mathbb{G}_1^2$ and $U_2 = (\upsilon_2 G, \upsilon_1 \upsilon_2 G) \in \mathbb{G}_1^2$. Get random $\nu_1, \nu_2 \xleftarrow{\$} \mathbb{Z}_q^*$, set $V_1 = (H, \nu_1 H) \in \mathbb{G}_2^2$ and $V_2 = (\nu_2 H, \nu_1 \nu_2 H) \in \mathbb{G}_2^2$.

Witness-indistinguishability string: Get random $\upsilon_1, \upsilon_2 \xleftarrow{\$} \mathbb{Z}_q^*$, set $U_1 = (G, \upsilon_1 G) \in \mathbb{G}_1^2$ and $U_2 = (\upsilon_2 G, \upsilon_1 (\upsilon_2 - 1) G) \in \mathbb{G}_1^2$. Get random $\nu_1, \nu_2 \xleftarrow{\$} \mathbb{Z}_q^*$, set $V_1 = (H, \nu_1 H) \in \mathbb{G}_2^2$ and $V_2 = (\nu_2 H, \nu_1 (\nu_2 - 1) H) \in \mathbb{G}_2^2$.

The CRS is defined as $\varsigma = (\vec{U} = (U_1, U_2), \vec{V} = (V_1, V_2))$.

We also define the following operations for $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$, and $z \in \mathbb{G}_T$:

$\iota_1 : \mathbb{G}_1 \to \mathbb{G}_1^2$ such that $\iota_1(P) = (\mathcal{O}, P)$.

$\iota_2 : \mathbb{G}_2 \to \mathbb{G}_2^2$ such that $\iota_2(Q) = (\mathcal{O}, Q)$.

$$\iota_T : \mathbb{G}_T \to \mathbb{G}_T^4 \text{ such that } \iota_T(z) = \begin{pmatrix} 1 & 1 \\ 1 & z \end{pmatrix}.$$

$$E : \mathbb{G}_1^2 \times \mathbb{G}_2^2 \to \mathbb{G}_T^4 \text{ such that } E((P_1, P_2), (Q_1, Q_2)) = \begin{pmatrix} e(P_1, Q_1) & e(P_1, Q_2) \\ e(P_2, Q_1) & e(P_2, Q_2) \end{pmatrix}.$$

The addition operation in either $\mathbb{G}_1^2$, $\mathbb{G}_2^2$ or $\mathbb{G}_T^4$ is computed component-wise, and the multiplication by a scalar number is computed distributing the scalar, resulting in the product of each component by the scalar number.

The implicit setup (i.e., the group operations and mappings used for all CRS instances) is defined as $gk = (\Lambda, \iota_1, \iota_2, \iota_T, E)$.

Figure 5 illustrates the *GSSetup* method.

- $GSCommit(gk, \varsigma, P\ [, r]) \to \kappa$

  From the setup gk and from the CRS ς, the value P is committed to κ with the random opening $r = (r_1, r_2) \xleftarrow{\$} \mathbb{Z}_q^2$ as follows:

  $C : \mathbb{G}_1 \times \mathbb{Z}_q^2 \to \mathbb{G}_1^2$ such that $C(P, r) = \iota_1(P) + r_1 U_1 + r_2 U_2$.

  $D : \mathbb{G}_2 \times \mathbb{Z}_q^2 \to \mathbb{G}_2^2$ such that $D(P, r) = \iota_2(P) + r_1 V_1 + r_2 V_2$.

Figure 5: GSProof: Setup protocol

Setup server $\mathcal{G}(\mathbf{1}^k, \mathbf{\Lambda} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H, q))$

$\upsilon_1, \upsilon_2, \nu_1, \nu_2 \xleftarrow{\$} \mathbb{Z}_q$

$\vec{U} = ((G, \upsilon_1 G), (\upsilon_2 G, \upsilon_1 \upsilon_2 G))$

$\vec{V} = ((H, \nu_1 H), (\nu_2 H, \nu_1 \nu_2 H))$

$\varsigma = (\vec{U}, \vec{V})$

$\iota_1 : \mathbb{G}_1 \to \mathbb{G}_1^2$ such that $\iota_1(P) = (O, P)$

$\iota_2 : \mathbb{G}_2 \to \mathbb{G}_2^2$ such that $\iota_2(Q) = (O, Q)$

$\iota_T : \mathbb{G}_T \to \mathbb{G}_T^4$ such that $\iota_T(z) = \begin{pmatrix} 1 & 1 \\ 1 & z \end{pmatrix}$

$E : \mathbb{G}_1^2 \times \mathbb{G}_2^2 \to \mathbb{G}_T^4$ such that $E((P_1, P_2), (Q_1, Q_2)) = \begin{pmatrix} e(P_1, Q_1) & e(P_1, Q_2) \\ e(P_2, Q_1) & e(P_2, Q_2) \end{pmatrix}$

$gk = (\Lambda, \iota_1, \iota_2, \iota_T, E)$

$\implies (gk, \varsigma)$

Source: Author

The commitment is $\kappa = \begin{cases} C(P, r) & \text{if } P \in \mathbb{G}_1 \\ D(P, r) & \text{if } P \in \mathbb{G}_2 \end{cases}$.

Figure 6 illustrates the *GSCommit* method.

Figure 6: GSProof: Commit protocol

Prover $\mathcal{P}(gk, \varsigma, P\ [, r])$

$r = (r_1, r_2) \xleftarrow{\$} \mathbb{Z}_q^2$

If $P \in \mathbb{G}_1$:

$\kappa = \iota_1(P) + r_1 U_1 + r_2 U_2$

Else (if $P \in \mathbb{G}_2$):

$\kappa = \iota_2(P) + r_1 V_1 + r_2 V_2$

$\implies \kappa$

Source: Author

- $GSProve(gk, \varsigma, \{eq_k\}_{i=k...N}, \vec{X}, \vec{Y}\ [, \vec{R}]\ [, \vec{S}]) \to \vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1...N}$

  Commit all variables $X_i \in \vec{X} \subset \mathbb{G}_1^m$ (resp. $Y_j \in \vec{Y} \subset \mathbb{G}_2^n$) with random opening $r_i \in \vec{R} \subset Mat_{m\times 2}(\mathbb{Z}_q)$ (resp. $s_j \in \vec{S} \subset Mat_{n\times 2}(\mathbb{Z}_q)$) as $C_i \leftarrow GSCommit(gk, \varsigma, X_i, r_i)$ (resp. $D_j \leftarrow GSCommit(gk, \varsigma, Y_j, s_j)$), representing them as $\vec{C} = \{C_i\}_{i=1...m} \in (\mathbb{G}_1^2)^m$ (resp. $\vec{D} = \{D_j\}_{j=1...n} \in (\mathbb{G}_2^2)^n$).

  For each equation $eq_k := \prod_{j=1}^{n} e(A_j, Y_j) \prod_{i=1}^{m} e(X_i, B_i) \prod_{i=1}^{m} \prod_{j=1}^{n} e(X_i, Y_j)^{\delta_{(i,j)}} = c$

and random $\Upsilon \xleftarrow{\$} Mat_{2\times2}(\mathbb{Z}_q)$, we represent $\delta_{(i,j)} \in \Delta \subset Mat_{m\times n}$, $A_j \in \vec{A} \subset \mathbb{G}_1^n$ and $B_i \in \vec{B} \subset \mathbb{G}_2^m$. The relation $\iota_i : \mathbb{G}_i^n \rightarrow (\mathbb{G}_i^m)^n$, for $i = \{1, 2, T\}$, is defined as $\forall x_j \in \vec{x} : \iota_i(\vec{x}) = \{\iota_i(x_j)\}_{j=1\ldots n}$.

The proof is $\begin{cases} \varphi_k = \vec{S}\,\iota_1(\vec{A}) + \vec{S}\,\Delta^T\,\iota_1(\vec{X}) + \Upsilon\,\vec{U} \\ \pi_k = \vec{R}^T\,\iota_2(\vec{B}) + \vec{R}^T\,\Delta\,\iota_2(\vec{Y}) + \left(\vec{R}^T\,\Delta\,\vec{S} - \Upsilon^T\right)\vec{V} \end{cases}$.

Output also the auxiliary commitments $\vec{C}$ and $\vec{D}$.

Figure 7 illustrates the *GSProve* method.

Figure 7: GSProof: Prove protocol

Prover $\mathcal{P}(gk, \varsigma, \{eq_k\}_{i=k\ldots N}, \vec{X}, \vec{Y})$

$eq_k := \prod_{j=1}^{n} e(A_j, Y_j) \prod_{i=1}^{m} e(X_i, B_i) \prod_{i=1}^{m} \prod_{j=1}^{n} e(X_i, Y_j)^{\delta_{(i,j)}} = c$

$R \xleftarrow{\$} Mat_{m\times2}(\mathbb{Z}_q) \quad S \xleftarrow{\$} Mat_{2\times n}(\mathbb{Z}_q) \quad \Upsilon \xleftarrow{\$} Mat_{2\times2}(\mathbb{Z}_q)$

$\forall i \in [1, m] : C_i \leftarrow GSCommit(gk, \varsigma, X_i, R_i)$

$\forall i \in [1, m] : \vec{X}_{I,i} \leftarrow \iota_1(X_i) \quad \vec{A}_{I,i} \leftarrow \iota_1(A_i)$

$\forall j \in [1, n] : D_j \leftarrow GSCommit(gk, \varsigma, Y_j, S_j^T)$

$\forall j \in [1, n] : \vec{Y}_{I,j} \leftarrow \iota_2(Y_j) \quad \vec{B}_{I,j} \leftarrow \iota_2(B_j)$

For each $k \in [1, N]$:

$\varphi_k = \vec{S}\,\vec{A}_I + \vec{S}\,\Delta^T\,\vec{X}_I + \Upsilon\,\vec{U}$

$\pi_k = \vec{R}^T\,\vec{B}_I + \vec{R}^T\,\Delta\,\vec{Y}_I + \left(\vec{R}^T\,\Delta\,\vec{S} - \Upsilon^T\right)\vec{V}$

$\implies (\vec{C}, \vec{D}, \{\varphi_k, \pi_k\}_{k=1\ldots N})$

Source: Author

- $GSVerify(gk, \varsigma, \{eq_k\}_{k=1\ldots N}, \vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1\ldots N}) \rightarrow \{0, 1\}$

  Output 1 if and only if for all equations $\{eq_k\}_{k=1\ldots N}$ the verification equation holds: $E(\iota_1(\vec{A}), \vec{D}) + E(\vec{C}, \iota_2(\vec{B})) + E(\vec{C}, \Delta\,\vec{D}) = \iota_T(c) + E(\vec{U}, \pi_k) + E(\varphi_k, \vec{V})$.

Figure 8 illustrates the *GSVerify* method.

3.7 Verifiable Random Function

A verifiable random function (VRF) is a especial type of pseudorandom function that allows anyone who knows the secret seed to evaluate the result at some point to

Figure 8: GSProof: Verify protocol

Verifier $\mathcal{V}(gk, \varsigma, , \{eq_k\}_{i=k\ldots N}, \vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1\ldots N})$

$eq_k := \prod_{j=1}^{n} e(A_j, Y_j) \prod_{i=1}^{m} e(X_i, B_i) \prod_{i=1}^{m} \prod_{j=1}^{n} e(X_i, Y_j)^{\delta_{(i,j)}} = c$

$\forall i \in [1, m] : \vec{A}_{I,i} \leftarrow \iota_1(A_i)$

$\forall j \in [1, n] : \vec{B}_{I,j} \leftarrow \iota_2(B_j)$

For each $k \in [1, N]$:

$$\implies E(\vec{A_I}, \vec{D}) + E(\vec{C}, \vec{B_I}) + E(\vec{C}, \Delta\vec{D}) \stackrel{?}{=} \iota_T(c) + E(\vec{U}, \pi_k) + E(\varphi_k, \vec{V})$$

Source: Author

a value indistinguishable from a number sampled at random from some distribution. It also allows the evaluator to produce a proof that the computation is indeed correct without compromising the unpredictability of the evaluation at any other point (MICALI; RABIN; VADHAN, 1999). More formally, it is defined as follows:

Definition 26 (Verifiable random function (MICALI; RABIN; VADHAN, 1999))**.** *A set of algorithms* $(\mathcal{G}, \mathcal{F}, \mathcal{P}, \mathcal{V})$*, with function generator* $\mathcal{G}$*, function evaluator* $\mathcal{F}$*, prover* $\mathcal{P}$ *and verifier* $\mathcal{V}$*, is a* verifiable random function *if the following properties hold:*

Domain-range correctness: if the evaluation's input (seed and chosen point) is in valid range, then its output will also be in valid range with overwhelming probability.

Complete provability: if an honest prover outputs a valid proof, the honest verifier accepts this proof with overwhelming probability.

Unique provability: if a dishonest prover could generate two proofs for different evaluations at the same point, the honest verifier would accept both proofs with negligible probability (i.e., an honest prover cannot evaluate the same point to two distinct evaluations).

Residual pseudorandomness: the probability that an adversary is able to guess the evaluation at some point that has not been queried yet is negligible.

The algorithms proposed to these methods are the following:

- $VRFSetup(1^k\ [, s]) \rightarrow (sk, pk)$

  Creates the function's secret key sk from some random seed s, as well as its public key pk, given the security parameter k.

- $VRFEval(sk, x) \rightarrow Y$

  Evaluates the function to Y using the secret key sk at some point x.

- $VRFProve(sk, x) \rightarrow \phi_Y$

  Evaluates the function to Y and generates the proof ϕ_Y that the evaluation is correct with respect to the secret key sk and the point x.

- $VRFVerify(pk, x, Y, \phi_Y) \rightarrow \{0, 1\}$

  Verifies if the proof ϕ_Y was correctly constructed, with regard to the evaluation Y at point x, using the public key pk.

Of especial interest to this work are two VRF instantiations described in (BELENKIY et al., 2009), in which the verification of x uses a PPE, so knowledge of s and x can be proved by the Groth-Sahai method. The first instantiation, presented in Section 3.7.1, is a simulatable (i.e., zero-knowledge provable) VRF, which is used in the proposed scheme to generate a serial number and a proof of ownership of the digital card having this serial number. The second one, presented in Section 3.7.2, receives additional parameters that are used to generate a transference tag number, which allows illegal duplications to be identified, both in e-cash protocols and in the proposed trade carding protocol. Both are secure under q-DDHI assumption (in $\mathbb{G}_1$) and SXDH (for the Groth-Sahai proof).

3.7.1 VRF instantiation 1

In what follows, we present a concrete instantiation of a VRF secure under q-DDHI assumption in $\mathbb{G}_1$, with proofs secure under the SXDH assumption. As previously

mentioned, this instantiation allows the generation of a card's serial number and the proof of ownership. The methods *VRFSetup*, *VRFEval*, *VRFProve* and *VRFVerify* are presented in Figures 9, 10, 11 and 12, respectively.

- $VRFSetup(1^k\ [, s\ [, open_s]]) \rightarrow (sk, pk)$

  For security parameter k, consider the asymmetric bilinear pairing setting $\Lambda = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H, q)$ in the setup $(gk, \varsigma) \leftarrow GSSetup(1^k)$ of a Groth-Sahai instantiation secure under SXDH assumption. Get random seed $s \xleftarrow{\$} \mathbb{Z}_q$ and a random opening $open_s \xleftarrow{\$} \mathbb{Z}_q^2$ of a Groth-Sahai commitment. Set the private input $sk = (s, open_s)$ and the public input $pk \leftarrow GSCommit(gk, \varsigma, sH, open_s)$.

- $VRFEval(sk, x) \rightarrow Y$

  Given the private input sk and some offset x, compute and output $Y = \frac{1}{s+x} G$.

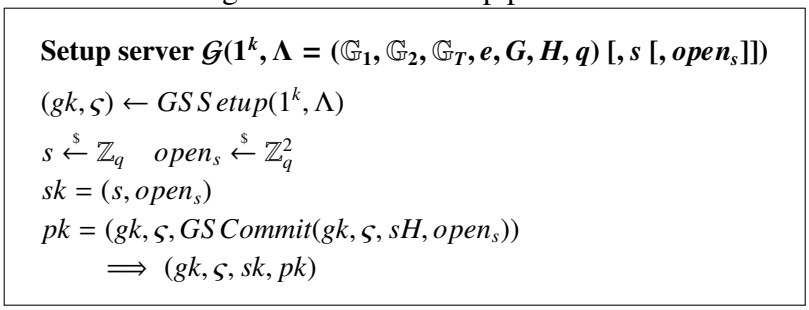
- $VRFProve(sk, x) \rightarrow \phi_Y$

  Given the private input sk and the offset x of an evaluation $Y \leftarrow VRFEval(sk, x)$, set $Y' = Y$, $s' = s$ and $x' = x$. Consider equations $eq = \{e(Y, 1H)\ e(Y', 1H)^{-1} = 1; e(1G, sH)\ e(1G, s'H)^{-1} = 1; e(1G, xH)\ e(1G, x'H)^{-1} = 1; e(Y', s'H)\ e(Y', x'H) = e(G, H); e(1G, H) = e(G, H); e(G, 1H) = e(G, H)\}$, with variables $\vec{X} = \{1G, Y'\}$ in $\mathbb{G}_1$ and $\vec{Y} = \{1H, sH, s'H, xH, x'H\}$ in $\mathbb{G}_2$. Create proof $\phi_Y = (\vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1...6}) \leftarrow GSProve(gk, \varsigma, eq, \vec{X}, \vec{Y}, \{\_, \_\}, \{\_, open_s, \_, \_, \_\})$ (where the symbol _ represent an empty input).

- $VRFVerify(pk, Y, \phi_Y) \rightarrow \{0, 1\}$

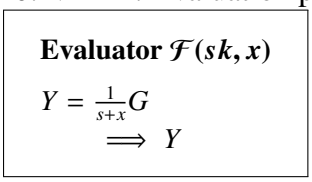
  Parse $\phi_Y = (\vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1...6})$ and consider the equations $eq = \{e(Y, 1H)\, e(Y', 1H)^{-1} = 1; e(1G, sH)\, e(1G, s'H)^{-1} = 1; e(1G, xH)\, e(1G, x'H)^{-1} = 1; e(Y', s'H)\ e(Y', x'H) = e(G, H); e(1G, H) = e(G, H); e(G, 1H) = e(G, H)\}$. Output $GSVerify(gk, \varsigma, eq, \vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1...6})$.

Figure 9: VRF 1: Setup protocol



Source: Author

Figure 10: VRF 1: Evaluation protocol



Source: Author

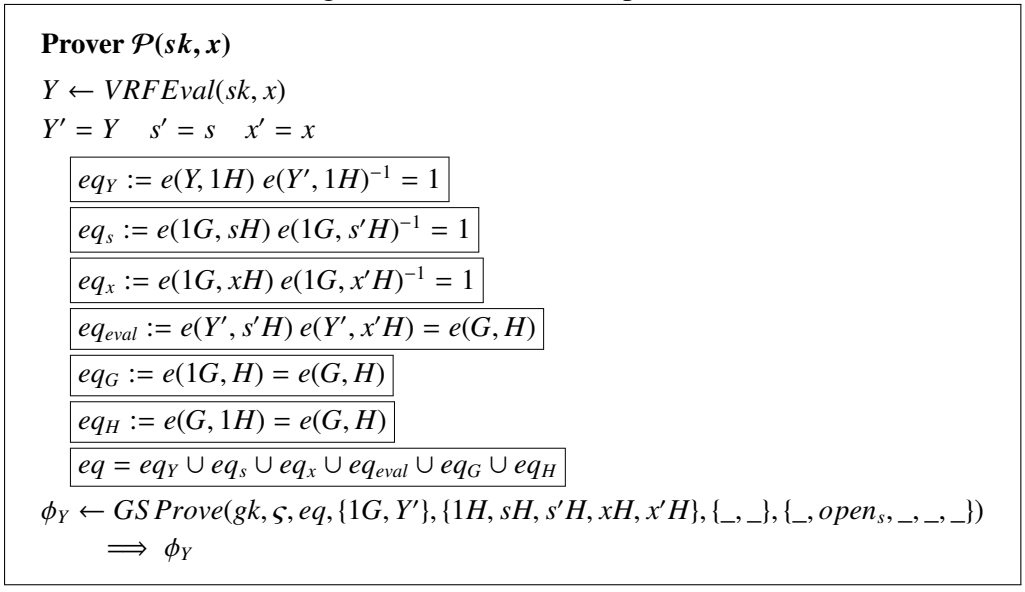
When compared to Definition 26, the method $VRFVerify$ has a slight difference in its parameters because the proof is zero-knowledge, so the offset x is kept secret even when proving the evaluation.

In this proof, we need 2 commitments in $\mathbb{G}_1$ ($1G, Y'$) and 5 commitments in $\mathbb{G}_2$ ($1H, sH, s'H, xH, x'H$), resulting in 4 elements in $\mathbb{G}_1$ and 10 elements in $\mathbb{G}_2$. We also have 6 equations to prove, which correspond to 24 extra elements in $\mathbb{G}_1$ and 24 in $\mathbb{G}_2$. Therefore, the proof ϕ_Y has 28 elements in $\mathbb{G}_1$ and 34 elements in $\mathbb{G}_2$, and the evaluation Y has 1 extra element in $\mathbb{G}_1$.

3.7.2 VRF instantiation 2

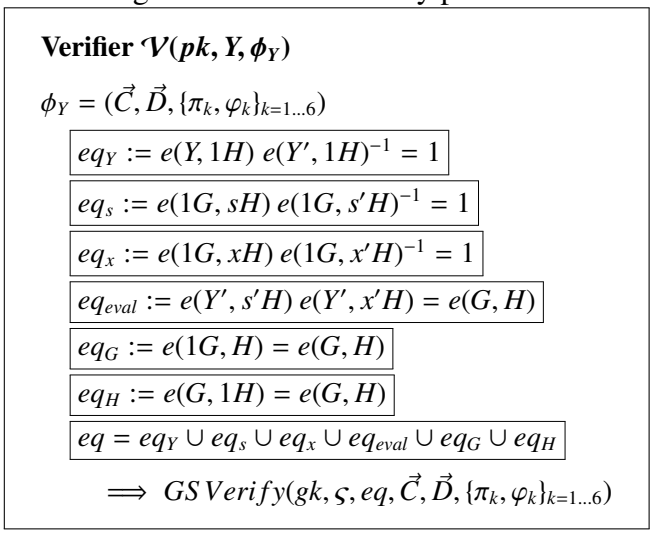
Now we present the adapted instantiation of a VRF secure under q-DDHI assumption in $\mathbb{G}_1$, with proofs secure under SXDH assumption, which is employed in the proposed scheme for creating transference tag numbers. The methods *VRF2Setup*, *VRF2Eval*, *VRF2Prove* and *VRF2Verify* are presented in Figures 13, 14, 15 and 16, respectively.

Figure 11: VRF 1: Prove protocol



Source: Author

Figure 12: VRF 1: Verify protocol



Source: Author

- $VRF2Setup(1^k\ [, s\ [, open_s]]) \rightarrow (sk, pk)$

  Just like instantiation 1, output $VRFSetup(1^k\ [, s\ [open_s]])$.

- $VRF2Eval(sk, x, w, R) \rightarrow T$

  Given the private input sk, some offset x, and two factors, one public R and one private w, compute and output $T = (G^R)^w \frac{1}{s+x} G$.

- $VRF2Prove(sk, x, w, R) \rightarrow \phi_T$

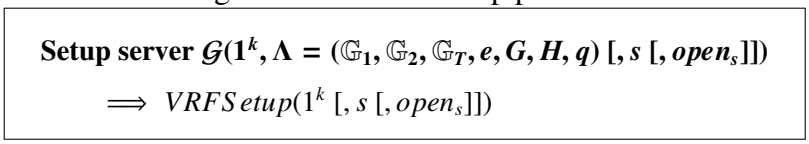
  Given the private input sk and the offset x of an evaluation from instantiation 1, $Y \leftarrow VRFEval(sk, x)$, and the factors R and w of an evaluation from this instance, $T \leftarrow VRF2Eval(sk, x, w, R)$, set $T' = T$, $s' = s$, $x' = x$ and $w'' = w' = w$. Consider equations $eq = \{e(T, 1H)e(T', 1H)^{-1} = 1; e(1G, sH)e(1G, s'H)^{-1} = 1; e(1G, xH)e(1G, x'H)^{-1} = 1; e(1G, wH)\, e(1G, w'H)^{-1} = 1; e(w''G, H)\, e(-G, w'H) = 1; e(Y, s'H)\, e(Y, x'H) = e(G, H); e(1G, H) = e(G, H); e(G, 1H) = e(G, H)\}$, with variables $\vec{X} = \{1G, Y, w''G, T'\}$ in $\mathbb{G}_1$ and $\vec{Y} = \{1H, sH, s'H, xH, x'H, wH, w'H\}$ in $\mathbb{G}_2$. Create proof $\phi_T = (\vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1...8}) = GSProve(gk, \varsigma, eq, \vec{X}, \vec{Y}, \{\_, \_, \_, (open_{w''G}\, R + open_Y)\}, \{\_, open_s, \_, \_, \_, \_, \_\})$ (where the symbol _ represents an empty input, and $open_{w''G}$ and $open_Y$ are openings produced by $GSProof$ for the variables $w''G$ and Y, respectively).

- $VRF2Verify(pk, R, T, \phi_T) \rightarrow \{0, 1\}$

  Parse $\phi_T = (\vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1...8})$ and consider the equations $eq = \{e(T, 1H)\, e(T', 1H)^{-1} = 1; e(1G, sH)e(1G, s'H)^{-1} = 1; e(1G, xH)e(1G, x'H)^{-1} = 1; e(1G, wH)\, e(1G, w'H)^{-1} = 1; e(w''G, H)\, e(-G, w'H) = 1; e(Y, s'H)\, e(Y, x'H) = e(G, H); e(1G, H) = e(G, H); e(G, 1H) = e(G, H)\}$. Get $C_{w''G}$, C_Y and C_T from $\vec{C}$, and, if $C_T \neq C_{w''G}\, R + C_Y$, output 0; otherwise, output $GSVerify(gk, \varsigma, eq, \vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1...8})$.

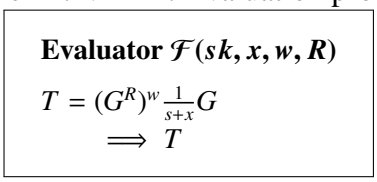
Similarly to instantiation 1, the method $VRF2Verify$ has a slight difference in its

Figure 13: VRF 2: Setup protocol



Source: Author

Figure 14: VRF 2: Evaluation protocol



Source: Author

parameters when compared to Definition 26 because the proof is zero-knowledge, so the offset x is kept secret even when proving the evaluation.

In this proof, we need 4 commitments in $\mathbb{G}_1$ ($1G, Y, w''G, T$) and 7 commitments in $\mathbb{G}_2$ ($1H, sH, s'H, xH, x'H, wH, w'H$), resulting in 8 elements in $\mathbb{G}_1$ and 14 elements in $\mathbb{G}_2$. We also have 8 equations to prove, which leads to additional 32 elements in $\mathbb{G}_1$ and 32 in $\mathbb{G}_2$. As a result, the proof ϕ_T has 40 elements in $\mathbb{G}_1$ and 46 elements in $\mathbb{G}_2$, and the evaluation T has 1 extra element in $\mathbb{G}_1$.

3.8 Provable blind signature

Blind signatures were originally proposed in the context of anonymous e-cash (CHAUM, 1983), allowing a user to obtain a valid signature on values unknown to the signer. If transferability is required, the user doing the transfer also needs to prove knowledge of the signed values.

Two important classes of provable signatures are Camenisch-Lysyanskaya (CL) signatures (CAMENISCH; LYSYANSKAYA, 2003) and structure-preserving (or automorphic) signatures (ABE et al., 2010). Structure-preserving signatures are far more important to transferable e-cash since their protocols were built to use non-interactive proofs of knowledge (Groth-Sahai proofs), while CL-signatures use interactive ones.

Figure 15: VRF 2: Prove protocol

Prover $\mathcal{P}(sk, x, w, R)$

$Y \leftarrow VRFEval(sk, x) \quad T \leftarrow VEF2Eval(sk, x, w, R)$

$Y' = Y \quad T' = T \quad s' = s \quad x' = x \quad w' = w$

$\boxed{eq_T := e(T, 1H)\, e(T', 1H)^{-1} = 1}$

$\boxed{eq_s := e(1G, sH)\, e(1G, s'H)^{-1} = 1}$

$\boxed{eq_x := e(1G, xH)\, e(1G, x'H)^{-1} = 1}$

$\boxed{eq_w := e(1G, wH)\, e(1G, w'H)^{-1} = 1}$

$\boxed{eq_{w'} := e(w''G, H)\, e(-G, w'H) = 1}$

$\boxed{eq_{eval} := e(Y, s'H)\, e(Y, x'H) = e(G, H)}$

$\boxed{eq_G := e(1G, H) = e(G, H)}$

$\boxed{eq_H := e(G, 1H) = e(G, H)}$

$\boxed{eq = eq_T \cup eq_s \cup eq_x \cup eq_w \cup eq_{w'} \cup eq_{eval} \cup eq_G \cup eq_H}$

$\vec{X} = \{1G, Y, w''G, T'\} \quad \vec{R} = \{\_, \_, \_, (open_{w''G}\, R + open_Y)\}$

$\vec{Y} = \{1H, sH, s'H, xH, x'H, wH, w'H\} \quad \vec{S} = \{\_, open_s, \_, \_, \_, \_, \_\}$

$\phi_T \leftarrow GS\,Prove(gk, \varsigma, eq, \vec{X}, \vec{Y}, \vec{R}, \vec{S})$

$\quad \Longrightarrow \phi_T$

Source: Author

Figure 16: VRF 2: Verify protocol

Verifier $\mathcal{V}(pk, T, \phi_T)$

$\phi_Y = (\vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1\ldots6})$

$\boxed{eq_T := e(T, 1H)\, e(T', 1H)^{-1} = 1}$

$\boxed{eq_s := e(1G, sH)\, e(1G, s'H)^{-1} = 1}$

$\boxed{eq_x := e(1G, xH)\, e(1G, x'H)^{-1} = 1}$

$\boxed{eq_w := e(1G, wH)\, e(1G, w'H)^{-1} = 1}$

$\boxed{eq_{w'} := e(w''G, H)\, e(-G, w'H) = 1}$

$\boxed{eq_{eval} := e(Y, s'H)\, e(Y, x'H) = e(G, H)}$

$\boxed{eq_G := e(1G, H) = e(G, H)}$

$\boxed{eq_H := e(G, 1H) = e(G, H)}$

$\boxed{eq = eq_T \cup eq_s \cup eq_x \cup eq_w \cup eq_{w'} \cup eq_{eval} \cup eq_G \cup eq_H}$

$\quad \Longrightarrow (C_T \stackrel{?}{=} C_{w''G}R + C_Y) \wedge GS\,Verify(gk, \varsigma, eq, \vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1\ldots6})$

Source: Author

The most efficient CL-signature schemes are built in RSA groups or in the target group of pairings (CAMENISCH; LYSYANSKAYA, 2003; CAMENISCH; LYSYANSKAYA, 2004). Even though we can use the Fiat-Shamir heuristic to transform interactive proofs into non-interactive ones, the arbitrary size of the input (the size of the message vector) does not guarantee security using function ensembles for implementation of the random oracle (CANETTI; GOLDREICH; HALEVI, 2004; GOLDWASSER; KALAI, 2003), and the original signature schemes do not prove security of non-interactive proofs.

Although structure-preserving signatures can be proved, a P-signature scheme (BELENKIY et al., 2008) (signatures with efficient *P*rotocols) offer stronger protocols for proofs. More specifically, a P-signature scheme applies three efficient protocols: (1) an interactive protocol to obtain a blind signature, (2) a non-interactive protocol to prove (and verify) knowledge of a signature, and (3) a non-interactive protocol to prove (and verify) that two commitments hide the same values.

For the purposes of this work, we adapt the P-signature scheme proposed by (IZABACHÈNE; LIBERT; VERGNAUD, 2011), converting it to an asymmetric pairing setting by means of the method proposed in (ABE et al., 2014). The reason for this modification is that, even though (IZABACHÈNE; LIBERT; VERGNAUD, 2011) is quite efficient, it uses symmetric pairing and supersingular elliptic curves, requiring fields of larger size to achieve a security level similar to what can be obtained with an asymmetric pairing (BARBULESCU et al., 2014). A concrete instantiation of the adapted P-signature is presented in Section 3.8.1.

3.8.1 P-signature instantiation

We now present the adaptation of the protocol proposed by Izabachène, Libert and Vergnaud (2011) converted to an asymmetric pairing setting. For conciseness, we do not show the conversion details here, leaving the step by step following the method

Table 5: Number of elements from each group when signing n messages

Object	$\mathbb{Z}_q$	$\mathbb{G}_1$	$\mathbb{G}_2$	*Object*	$\mathbb{Z}_q$	$\mathbb{G}_1$	$\mathbb{G}_2$
Private key (sk)	3	0	0	Opening (r)	1	0	0
Public key (pk)	0	$3+n$	$2+n$	Signature (σ)	1	5	1
Message ($\vec{m}$)	n	0	0	Proof of commitment (ϕ_K)	0	$2+8n$	4
Commitment (K)	0	1	0	Proof of signature (ϕ_σ)	0	$20+4n$	12

Source: Author

proposed by Abe et al. (2014) to Appendix A. The resulting protocol is secure under the following assuming the hardness of the following computational problems mentioned in Section 3.5: co-q-HSDH^{+1}, co-Flex-DH^{+3} and co-n-FlexDHE^{+2n}. For convenience of the reader, Table 5 lists the number of elements necessary for the signature scheme when signing n messages.

The methods *PSetup*, *PKeyGen*, *PCommit*, *PUpdateComm*, *PSign*, *PVerifySig*, *PWitGen*, *PVerifyWit*, *PProveCom*, *PVerifyProofCom*, *PObtainSig/PIssueSig*, *PProveSig* and *PVerifyProofSig* are presented, in order, in Figures 17 to 29.

- $PSetup(1^k) \rightarrow pparams$: For the security parameter k, consider the asymmetric bilinear pairing setting $\Lambda = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H, q)$ in the setup $(gk, \varsigma) \leftarrow GSSetup(1^k)$ of Groth-Sahai instantiation secure under SXDH assumption. For the sake of simplicity, these parameters $params = (gk, \varsigma)$ are omitted in the descriptions of the remainder operations.

- $PKeyGen(n) \rightarrow (pk, sk)$: Choose random $\alpha, \beta, \gamma, \omega \xleftarrow{\$} \mathbb{Z}_q^*$ and $U, U_0 \xleftarrow{\$} \mathbb{G}_1$. Compute $U_1 = \beta H, \Omega = \omega H, A = \gamma G$, as well as $\forall j \in [1, 2n] \setminus (n+1) : G_j = \alpha^j G, H_j = \alpha^j H$, to sign n messages. Output the private key $sk = (\gamma, \omega, \beta)$ and the public key $pk = (U, U_0, U_1, \Omega, A, \{G_j\}_{j=1...n}, \{H_j\}_{j=1...n})$.

- $PSign(sk, \vec{m}) \rightarrow \sigma$: For $\vec{m} = (m_1, ..., m_n)$, pick random $r \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $K = rG + \sum_{j=1}^{n} m_j G_{n+1-j}$. Choose random $c \xleftarrow{\$} \mathbb{Z}_q^*$ and compute: $\sigma_1 = \frac{\gamma}{\omega + c} G$, $\sigma_2 = cH, \sigma_3 = cU, \sigma_4 = c(U_0 + \beta K), \sigma_5 = cK, \sigma_6 = K, \sigma_r = r$.
  Output the signature $\sigma = (\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \sigma_r)$.

- $PVerifySig(pk, \sigma, \vec{m}) \rightarrow \{0,1\}$: Return 1 if and only if the following verification equations hold: $e(A,H) = e(\sigma_1, \Omega + \sigma_2)$, $e(U,\sigma_2) = e(\sigma_3, H)$, $e(\sigma_4, H) = e(U_0, \sigma_2)\, e(\sigma_5, U_1)$, $e(\sigma_5, H) = e(\sigma_6, \sigma_2)$, and $\sigma_6 = \sigma_r G + \sum_{j=1}^{n} m_j G_{n+1-j}$.

- $PCommit(pk, \vec{m}) \rightarrow (K, r)$: Choose random opening $r \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $K = rG + \sum_{j=1}^{n} m_j G_{n+1-j}$. Output the commitment $comm = (K, r)$.

- $PUpdateComm(pk, \vec{m}, K) \rightarrow K'$: Compute and output the updated commitment $K' = K + \sum_{j=1}^{n} m_j G_{n+1-j}$.

- $PWitGen(pk, i, \vec{m}, K, r) \rightarrow W_i$: If K is a commitment to message $\vec{m}$ with opening r, compute and output the i-th witness $W_i = rG_i + \sum_{j=1;j\neq i}^{n} m_j G_{n+1+i-j}$.

- $PVerifyWit(pk, j, m_j, W_j, K) \rightarrow \{0,1\}$: Return 1 if and only if the following equation holds: $e(K, H_j) = e(G_n, H_1)^{m_j} \cdot e(W_j, H)$.

- $PProveCom(pk, \vec{m}, K, r) \rightarrow \phi_K$: Generate witnesses for each message committed, $\forall j \in [1,n] : W_j \leftarrow WitGen(pk, j, \vec{m}, K, r)$. Set equations $eq = \{\forall j \in [1,n] : e(K, -H_j)\, e(m_j G_1, H_n)\, e(W_j, H) = 1, e(m_j G, H_1)\, e(m_j G_1, -H) = 1, e(m_j G, H_{2n})\, e(m_j G_{2n}, -H) = 1\}$ for variables $\vec{X} = \{K, \forall j \in [1,n] : W_j, m_j G_1, m_j G, m_j G_{2n}\}$ and $\vec{Y} = \varnothing$. Generate and output a Groth-Sahai proof of knowledge $\phi_K = (\vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1\ldots 3n}) \leftarrow GSProve(gk, \varsigma, eq, \vec{X}, \vec{Y})$.

- $PVerifyProofCom(\phi_K) \rightarrow \{0,1\}$: Parse $\phi_K = (\vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1\ldots 3n})$ and output $GSVerify(gk, \varsigma, eq, \vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1\ldots 3n})$.

- $\Big(PObtainSig(pk, \vec{m}_P) \leftrightarrow PIssueSig(sk, \vec{m}_S)\Big) \rightarrow \sigma$:

  - The User commits the message $\vec{m}_P$ as $(K, r') \leftarrow PCommit(pk, \vec{m}_P)$, then sends K to the Signer with a proof of knowledge $\phi_K \leftarrow PProveCom(pk, \vec{m}_P, K, r')$ that the commitment is valid.

  - The Signer verifies the proof of knowledge $PVerifyProofCom(\phi_K)$, updates the commitment to $K' \leftarrow PUpdateCom(pk, \vec{m}_S, K)$, and blindly signs

the commitment with random seeds $c, r'' \xleftarrow{\$} \mathbb{Z}_q^*$ as: $\sigma_1 = \frac{\gamma}{\omega+c} G$, $\sigma_2 = cH$, $\sigma_3 = cU$, $\sigma_4 = c(U_0 + \beta(K' + r''G))$, $\sigma_5 = c(K' + r''G)$, $\sigma_6 = K' + r''G$, and $\sigma'_r = r''$ and sends $\sigma' = (\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \sigma'_r)$ to the User.

  - The User updates $\sigma_r = r' + \sigma'_r$ and outputs $\sigma = (\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \sigma_r)$.

- $PProveSig(pk, \vec{m}, \sigma) \rightarrow \phi_\sigma$: Parse signature $\sigma = (\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \sigma_r)$. Generate witnesses for each message signed $\forall j \in [1, n] : W_j \leftarrow WitGen(pk, j, \vec{m}, \sigma_6, \sigma_r)$. Set signature equation validation: $eq_\sigma = \{e(\sigma_1, \Omega)\, e(\sigma_1, \sigma_2)\, e(-A, H) = 1, e(U_0, \sigma_2)\, e(\sigma_5, U_1)\, e(\sigma_4, H) = 1, e(U, \sigma_2)\, e(\sigma_3, -H) = 1, e(\sigma_5, H)\, e(\sigma_6, \sigma_2)^{-1} = 1\}$, message pertinence validation: $eq_{\vec{m}} = \{\forall j \in [1, n] : e(\sigma_6, -H_j)\, e(m_j G_1, H_n)\, e(W_j, H) = 1, e(m_j G, H_1)\, e(m_j G_1, H) = 1, e(m_j G, H_{2n})\, e(m_j G_{2n}, -H) = 1\}$, and equality commitment validation: $eq_K = \{e(A, 1H)\, e(-A, 1H) = 1, e(G, 1H) = e(G, H)\}$, with variables $\vec{X} = \{-A, U, U_0, \sigma_1, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \forall j \in [1, n] : W_j, m_j G_1, m_j G, m_j G_{2n}\}$ and $\vec{Y} = \{U_1, \Omega, \sigma_2, 1H\}$.

  Generate and output a Groth-Sahai proof of knowledge $\phi_\sigma = (\vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1\ldots(6+3n)}) \leftarrow GSProve(gk, \varsigma, eq_\sigma \cup eq_m \cup eq_K, \vec{X}, \vec{Y})$.

- $PVerifyProofSig(\phi_\sigma) \rightarrow \{0, 1\}$: Parse $\phi_\sigma = (\vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1\ldots(6+3n)})$ and output $GSVerify(gk, \varsigma, \vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1\ldots(6+3n)})$.

Figure 17: P-Signature: Setup protocol

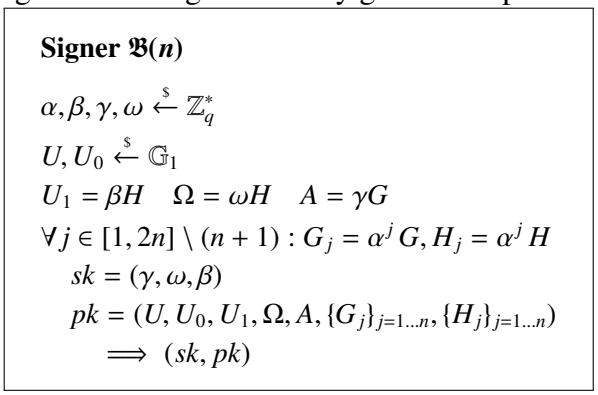
Setup server $\mathcal{G}(\mathbf{1}^k, \Lambda = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H, q))$

$(gk, \varsigma) \leftarrow GSSetup(1^k, \Lambda)$

$\Longrightarrow (gk, \varsigma)$

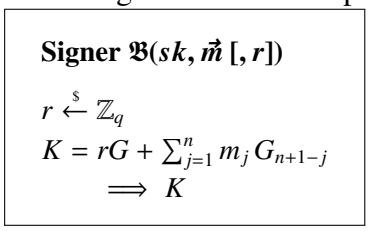
Source: Author

Figure 18: P-Signature: Key generation protocol



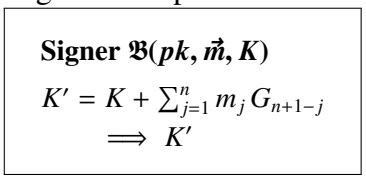
Source: Author

Figure 19: P-Signature: Commit protocol



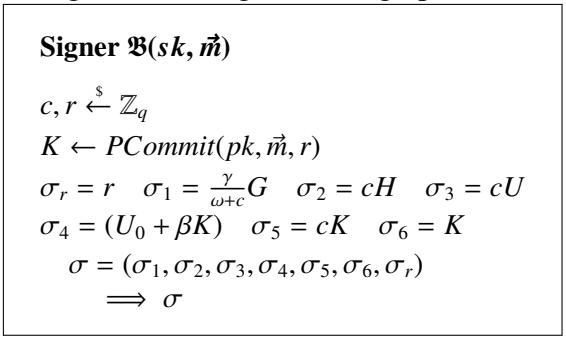
Source: Author

Figure 20: P-Signature: Update commitment protocol



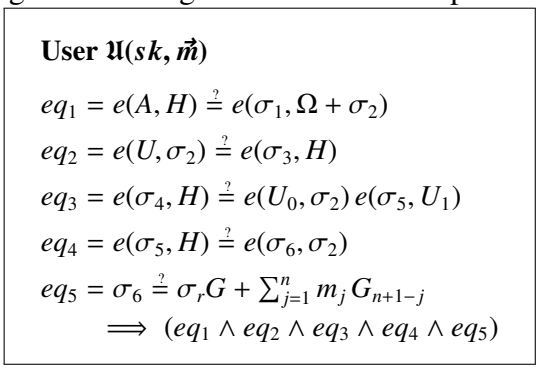
Source: Author

Figure 21: P-Signature: Sign protocol



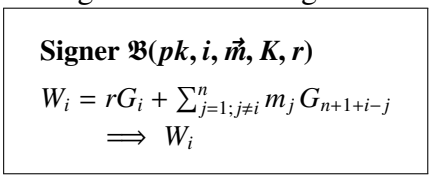
Source: Author

Figure 22: P-Signature: Verification protocol



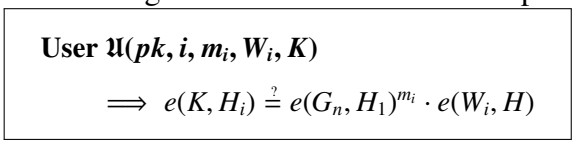
Source: Author

Figure 23: P-Signature: Witness generation protocol



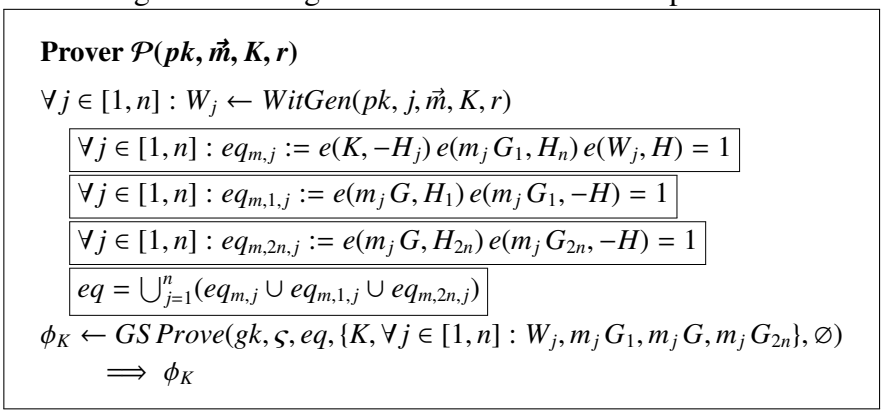
Source: Author

Figure 24: P-Signature: Witness verification protocol



Source: Author

Figure 25: P-Signature: Prove commitment protocol

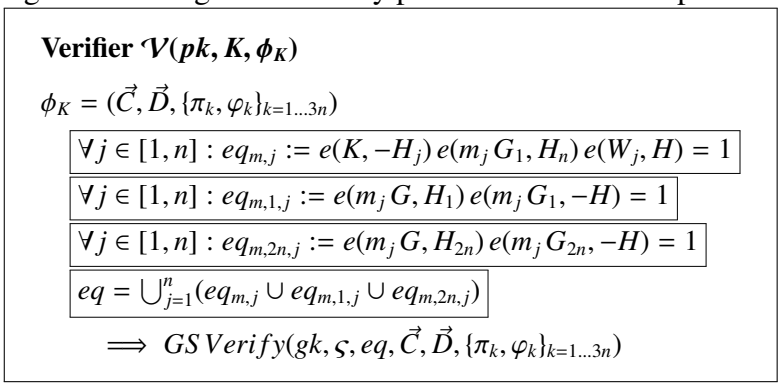


Source: Author

3.9 Compact e-cash

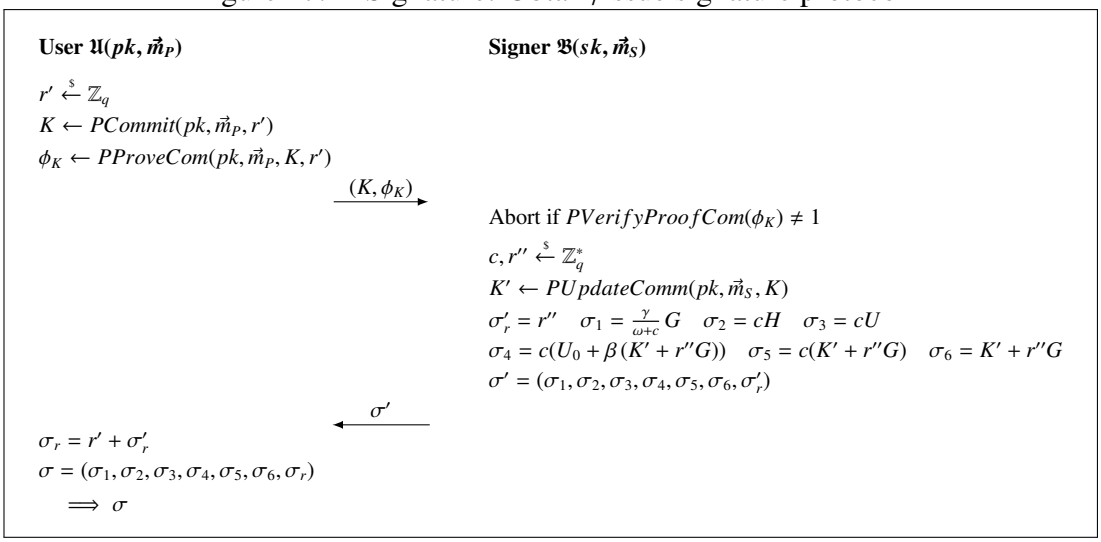
As electronic cash aims to emulate real cash, ensuring the result has strong security properties is of great interest for these protocols. Providing transferability is one

Figure 26: P-Signature: Verify proof of commitment protocol



Source: Author

Figure 27: P-Signature: Obtain/Issue signature protocol



Source: Author

of the most commonly discussed matters, since it is not possible to completely prevent double-spending (after all, the bits that represent a coin can be easily copied), but only ensure its detection. An online system in which a server intermediates every transaction can solve this problem by breaking the anonymity of the payment parties, the spender and the receiver, whenever a copy of a coin is duplicated. For an offline system, however, the sundry security protocols available in the literature provide different levels of anonymity, which can be informally defined as follows:

Definition 27 (Anonymity properties in e-cash (CANARD; GOUGET, 2008))**.** *An e-cash scheme can obtain one of the following anonymity properties:*

Figure 28: P-Signature: Prove signature protocol

Prover $\mathcal{P}(pk, \vec{m}, \sigma)$

$\forall j \in [1,n] : W_j \leftarrow WitGen(pk, j, \vec{m}, K, \sigma_r)$

$\boxed{eq_1 := e(\sigma_1, \Omega)\, e(\sigma_1, \sigma_2)\, e(-A, H) = 1}$

$\boxed{eq_2 := e(U_0, \sigma_2)\, e(\sigma_5, U_1)\, e(\sigma_4, H) = 1}$

$\boxed{eq_3 := e(U, \sigma_2)\, e(\sigma_3, -H) = 1}$

$\boxed{eq_4 := e(\sigma_5, H)\, e(\sigma_6, \sigma_2)^{-1} = 1}$

$\boxed{eq_A := e(A, 1H)\, e(-A, 1H) = 1}$

$\boxed{eq_H := e(G, 1H) = e(G, H)}$

$\boxed{\forall j \in [1,n] : eq_{m,j} := e(\sigma_6, -H_j)\, e(m_j G_1, H_n)\, e(W_j, H) = 1}$

$\boxed{\forall j \in [1,n] : eq_{m,1,j} := e(m_j G, H_1)\, e(m_j G_1, H) = 1}$

$\boxed{\forall j \in [1,n] : eq_{m,2n,j} := e(m_j G, H_{2n})\, e(m_j G_{2n}, -H) = 1}$

$\boxed{eq = eq_1 \cup eq_2 \cup eq_3 \cup eq_4 \cup eq_A \cup eq_H \cup \bigcup_{j=1}^{n} (eq_{m,j} \cup eq_{m,1,j} \cup eq_{m,2n,j})}$

$\vec{X} = \{-A, U, U_0, \sigma_1, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \forall j \in [1,n] : W_j, m_j G_1, m_j G, m_j G_{2n}\}$

$\vec{Y} = \{U_1, \Omega, \sigma_2, 1H\}$

$\phi_\sigma \leftarrow GS\,Prove(gk, \varsigma, eq, \vec{X}, \vec{Y})$

$\implies \phi_\sigma$

Source: Author

Figure 29: P-Signature: Verify proof of signature protocol

Verifier $\mathcal{V}(\phi_\sigma)$

$\phi_\sigma = (\vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1\ldots(6+3n)})$

$\boxed{eq_1 := e(\sigma_1, \Omega)\, e(\sigma_1, \sigma_2)\, e(-A, H) = 1}$

$\boxed{eq_2 := e(U_0, \sigma_2)\, e(\sigma_5, U_1)\, e(\sigma_4, H) = 1}$

$\boxed{eq_3 := e(U, \sigma_2)\, e(\sigma_3, -H) = 1}$

$\boxed{eq_4 := e(\sigma_5, H)\, e(\sigma_6, \sigma_2)^{-1} = 1}$

$\boxed{eq_A := e(A, 1H)\, e(-A, 1H) = 1}$

$\boxed{eq_H := e(G, 1H) = e(G, H)}$

$\boxed{\forall j \in [1,n] : eq_{m,j} := e(\sigma_6, -H_j)\, e(m_j G_1, H_n)\, e(W_j, H) = 1}$

$\boxed{\forall j \in [1,n] : eq_{m,1,j} := e(m_j G, H_1)\, e(m_j G_1, H) = 1}$

$\boxed{\forall j \in [1,n] : eq_{m,2n,j} := e(m_j G, H_{2n})\, e(m_j G_{2n}, -H) = 1}$

$\boxed{eq = eq_1 \cup eq_2 \cup eq_3 \cup eq_4 \cup eq_A \cup eq_H \cup \bigcup_{j=1}^{n} (eq_{m,j} \cup eq_{m,1,j} \cup eq_{m,2n,j})}$

$\implies GS\,Verify(gk, \varsigma, \vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1\ldots(6+3n)})$

Source: Author

Weak Anonymity (WA)*: An adversary cannot link a spending to a withdrawal. However, the adversary can find out if two spending operations were done by the same user.*

Strong Anonymity (SA)*: The scheme achieves WA and the adversary cannot distinguish if two spending operations were done by the same user. However, the adversary can recognize a coin that was observed in a previous spending.*

Full Anonymity (FA)*: The scheme achieves SA and the adversary cannot recognize a coin that was observed in a previous spending operation. However, the adversary can recognize a coin that he/she previously owned.*

Perfect Anonymity (PA)*: The scheme achieves FA and the adversary cannot decide whether or not he/she has owned a coin he/she received.*

Since it is proved that PA cannot be achieved if the bank is itself a possible adversary, two other properties can be defined by modifying this last property:

Spend-then-Observe (PA1)*: The scheme achieves SA and the adversary controlling the bank cannot link a coin he/she has previously possessed to one transferred between two honest users. However, the adversary can recognize a coin that he/she receives if he/she had previously owned it.*

Spend-then-Receive (PA2)*: The scheme achieves SA and the adversary cannot link a coin he/she has previously possessed to one transferred between two honest users. The adversary cannot control the bank.*

There is an inclusion relation among some of the properties (namely, PA $\Rightarrow$ *FA* $\Rightarrow$ *SA* $\Rightarrow$ *WA), but PA1 and PA2 do not satisfy FA.*

In the context of P2P TCGs, it is desirable to use a scheme that is both efficient and that provides the highest level of anonymity, thus protecting the players' privacy

without impairing usability. The compact e-cash scheme originally described in (CAMENISCH; HOHENBERGER; LYSYANSKAYA, 2005) and revised in (BELENKIY et al., 2009) is interesting for this purpose because (1) it allows several seed parameters (instead of coins) to be signed altogether and (2) it provides a direct method for identifying cheaters, who have their public key recovered, so the server does not need to screen the whole users' database in search for the culprit. Withdrawing several coins (i.e., a wallet) in a single message is not an essential property for a card game. It is an expensive operation, that increases the number of proofs of knowledge, and can be uncoupled from the protocol by reducing the size of the wallet to 1 coin. The protocol is also modular, allowing specific protocols to be replaced by more efficient ones whenever necessary. In addition, even though it only provides SA, in a physical TCG a player can also observe the cards being exchanged, which allows him/her to recognize that card in some future use; therefore, having FA or PA is not critical, as SA is enough to emulate quite accurately cart tradings in real life. In summary, building a TCG-oriented trading protocol over such e-cash scheme leads the following properties:

Correctness: When the bank and the users are honest, *Withdraw* will always succeed by generating a valid coin, *Spend* will succeed by the merchant accepting the received coin, the *Deposit* will succeed by accepting the deposited coin.

Strong anonymity: The bank, even in collusion with merchants, cannot link an execution of *Spend* with the execution of *Withdraw* that had generated the transfered coin.

Balance: No coalition of users is able to deposit more coins than the ones they withdrew without revoking the anonymity of at least one user of the coalition.

Identification of double-spenders: The bank can identify any user who has generated two valid coins with the same serial number.

Strong Exculpability: The bank, even in collusion with other users, cannot frame

a user that has never double-spent by generating two coins that identify the honest user. Furthermore, the transgressor is only responsible by the coins he/she had actually duplicated.

The original scheme is not transferable, but the version actually adopted in the proposed solution is based on the adaptation from (CANARD; GOUGET; TRAORÉ, 2008), which achieves transferability with strong anonymity, by means of the following operations (for a concrete instantiation and details, see (BELENKIY et al., 2009)):

- *Setup*: The bank generates a public/private key pair and publishes its public key together with the system's public parameters.
- *Register*: The user randomly generates a public/private key pair based on the system parameters and retrieves a certificate from the bank for the public key generated in this manner. The bank stores the user's identity and corresponding public keys, which allows users to be identified in case of double-spending.
- *Withdraw*: The user produces seed values and commits them to the bank, which in turn blindly signs those values. This creates a new anonymous wallet with as many coins as the number of seeds provided.
- *Spend*: Users may exchange either unspent coins from their wallets or coins previously received. In the former case, the user creates a new coin from the serial seed and treats it just like a received coin. Each time a coin is spent, a tag giving ownership of it to the receiver is added to the coin representation, making it grow in size. All tags must be verified by the receiver to ensure the previous transaction are valid and, thus, that the coin actually holds value.
- *Deposit*: The user sends the coin to the bank, which verifies if this coin had already been deposited. If it has, the bank verifies if this is a case of double-

deposit (i.e., if the user is trying to deposit the same coin twice) or of double-spending (i.e., if it was sent to two different users at some point in time).

- $Identify$: In case of double-spending, the bank retrieves the public key of the perpetrator, so the required administrative penalties can be applied.

Users may withdraw wallets with fixed L coins by obtaining a blind signature from the bank. It signs the seed parameter used to compute tags that identify the coin and verify if it was double-spent, the current owner to allow him/her to spend it, and the previous owners to identify the double-spender.

Each coin is identified by a serial number S, a random-like number to provide a unique identifier. It is then retrieved from the *serial number generation* function, a VRF which uses as input the index l of the coin, if more than one coin can be withdrawn (or the private key of the owner $l = sk_U$), and a seed s signed in the wallet. It is executed by the following algorithms:

- $\mathcal{F}_S(l, s) \rightarrow S$: For an implicit security level k, generate key-pair $(sk, pk) \leftarrow VRFSetup(1^k, l)$. Then, compute and output $S \leftarrow VRFEval(sk, s)$.

- $\mathcal{P}_S(l, s) \rightarrow \phi_S$: For an implicit security level k, generate key-pair $(sk, pk) \leftarrow VRFSetup(1^k, l)$. Then, compute and output $\phi_S \leftarrow VRFProve(sk, s)$.

- $\mathcal{V}_S(S, \phi_S) \rightarrow \{0, 1\}$: Parse the Groth-Sahai proof $\phi_S = (\vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1...6})$. Set $pk = C_{lH}$, retrieved from $\vec{D}$. Output $VRFVerify(pk, S, \phi_S)$.

The transference tag T identifies each transference, which allows the bank to identify a double-spender. It is concatenated to a list of transferences in the coin, thus allowing the bank to identify the transgressor, even if he/she is not the owner anymore. It is also retrieved from the *transference tag generation*, a modified version of the VRF (instantiation 2, Section 3.7.2) which inputs the private key of the owner sk_U, a seed

t signed in the wallet, and the hash of the private (that contains the owner) and public (e.g., a timestamp) information of the transference. It is executed by the following algorithms:

- $\mathcal{F}_T(sk_U, t, R) \rightarrow T$: For an implicit security level k, generate key-pair $(sk, pk) \leftarrow VRF2Setup(1^k, sk_U)$. Then, compute and output $T \leftarrow VRF2Eval(sk, t, sk_U, R)$.
- $\mathcal{P}_T(sk_U, t, R) \rightarrow \phi_T$: For an implicit security level k, generate key-pair $(sk, pk) \leftarrow VRF2Setup(1^k, sk_U)$. Then, compute and output $\phi_T \leftarrow VRF2Prove(sk, t, sk_U, R)$.
- $\mathcal{V}_T(R, T, \phi_T) \rightarrow \{0, 1\}$: Parse the Groth-Sahai proof $\phi_T = (\vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1...8})$. Set $pk = C_{sk_U H}$, retrieved from $\vec{D}$. Output $VRF2Verify(pk, R, T, \phi_T)$.

Finally, the ownership tag r is used to hide the private key of the coin's owner. Similarly to S, it is retrieved from the *ownership tag generation* function, a VRF which inputs the private key of the owner sk_U and some public information $\mathfrak{info}$ related to the transference. This tag is used to create the transference tag that allows the owner of the coin to prove that the last transference was directed to him/her, so this information is used to compute R, linking the transference tag T to the owner, represented by r. It is executed by the following algorithms:

- $\mathcal{F}_r(sk_U, \mathfrak{info}) \rightarrow r$: For an implicit security level k, generate key-pair $(sk, pk) \leftarrow VRFSetup(1^k, sk_U)$. Then, compute and output $r \leftarrow VRFEval(sk, \mathfrak{info})$.
- $\mathcal{P}_r(sk_U, \mathfrak{info}) \rightarrow \phi_r$: For an implicit security level k, generate key-pair $(sk, pk) \leftarrow VRFSetup(1^k, sk_U)$. Then, compute and output $\phi_r \leftarrow VRFProve(sk, \mathfrak{info})$.
- $\mathcal{V}_r(pk_U, r, \phi_r) \rightarrow \{0, 1\}$: Parse the Groth-Sahai proof $\phi_r = (\vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1...6})$. Set $pk = C_{sk_U H}$, retrieved from $\vec{D}$. Output $VRFVerify(pk, r, \phi_r) \wedge GSVerify(gk, \varsigma, \{e(1G, sk_U H) = pk_U; e(1G, H) = e(G, H)\}, \{1G\}, \{pk\})$.

We present a concrete instantiation of the protocol in Section 3.9.1

3.9.1 Compact e-cash instantiation

We now present the compact e-cash protocol proposed in (BELENKIY et al., 2009). This protocol is secure, given the security of the Groth-Sahai proof of knowledge, the VRFs and the P-signature. Figures 30, 31, 32, 33, 34 and 35 illustrate, respectively, the steps of the methods *Setup*, *Register*, *Withdraw*, *Spend*, *Deposit* and *Identify*.

- $Setup(1^k) \rightarrow (pparams, pk_R, pk_C, pk_L, \{\phi_{L,l}\}_{l=1...L})$: For the security parameter k, the central bank generates the system parameters of a signature setup $pparams = (gk, \varsigma) \leftarrow PSetup(1^k)$, which comprises the asymmetric pairing setting $\Lambda = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H, q)$ in the setup of a Groth-Sahai proof (gk, ς). These parameters are used by the subsequent methods and, for shortness, are omitted in their descriptions.

  It initializes an empty list of deposited coins $\mathcal{CS} = \varnothing$, and generates three key-pairs: $(sk_R, pk_R) \leftarrow PKeyGen(2)$ to register users, $(sk_C, pk_C) \leftarrow PKeyGen(3)$ to mint coins, and $(sk_L, pk_L) \leftarrow PKeyGen(1)$ to sign coin indexes. It produces $\forall l \in [1, L] : \sigma_l = PSign(sk_L, \{l\})$, and the proofs $\forall l \in [1, L] : \phi_{L,l} \leftarrow PProofSig(pk_L, \{l\}, \sigma_l)$. It then publishes $(pparams, pk_R, pk_C, pk_L, \{\phi_{L,l}\}_{l=1...L})$.

- $Register\Big(\mathfrak{U}(pk_R, id_U[, sk_U]) \leftrightarrow \mathfrak{B}(sk_R)\Big) \rightarrow \sigma_U$: User $\mathfrak{U}$ with identity id_U generates a secret key $sk_U \xleftarrow{\$} \mathbb{Z}_q$ and computes the public key $pk_U = e(G, sk_U H)$. $\mathfrak{U}$ generates a proof of knowledge $\phi_U = (\vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1,2}) \leftarrow GSProof\,(gk, \varsigma, \{e(1G, sk_U H) = pk_U; e(1G, H) = e(G, H)\}, \{1G\}, \{sk_U H\})$. The triple (id_U, pk_U, ϕ_U) is sent to the bank $\mathfrak{B}$.

  If the proof ϕ_U is valid by $GSVerify(gk, \varsigma, \{e(1G, sk_U H) = pk_U; e(1G, H) = e(G, H)\}, \vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1,2}) \stackrel{?}{=} 1$, $\mathfrak{B}$ generates a signa-

ture $\sigma_U \leftarrow PSign(sk_R, \{id_U, pk_U\})$ and sends it to $\mathfrak{U}$. If the verification $PVerifySig(pk_R, \sigma_U, \{id_U, pk_U\}) \stackrel{?}{=} 1$ is accepted, $\mathfrak{U}$ can then present σ_U as his/her certificate.

- $Withdraw\Big(\mathfrak{U}(sk_U, pk_C) \leftrightarrow \mathfrak{B}(sk_C, pk_C)\Big) \rightarrow \mathfrak{wallet}$: To withdraw a wallet with L coins, the user $\mathfrak{U}$ generates a partial identifier seed $s' \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ and a transference seed $t \stackrel{\$}{\leftarrow} \mathbb{Z}_q$, and the bank $\mathfrak{B}$ generates the coin's partial identifier component $s'' \stackrel{\$}{\leftarrow} \mathbb{Z}_q$. Both parties execute the interactive protocol to obtain a blind signature $\sigma_W \leftarrow \Big(PObtainSig(pk_C, \{sk_U, s', t\}) \leftrightarrow PIssueSig(sk_C, \{0, s'', 0\})\Big)$ that is returned to $\mathfrak{U}$ together with s''. $\mathfrak{U}$ stores the wallet $\mathfrak{wallet} = (s = s' + s'', t, L, \sigma_W)$.

- $Spend\Big(\mathfrak{U}_1(sk_{U_1}, pk_{U_2}, \{\mathfrak{wallet}|\mathfrak{coin}\}) \leftrightarrow \mathfrak{U}_2(sk_{U_2}, pk_{U_1})\Big) \rightarrow (\mathfrak{coin}'[, \mathfrak{wallet}'])$: The receiver $\mathfrak{U}_2$ chooses some public information $\mathfrak{info} \leftarrow \{0,1\}^*$ (e.g., a timestamp) and computes $r \leftarrow \mathcal{F}_r(sk_{U_2}, \mathcal{H}(\mathfrak{info}))$ and a proof of validity $\phi_r \leftarrow \mathcal{P}_r(sk_{U_2}, \mathcal{H}(\mathfrak{info}))$. $\mathfrak{U}_2$ then sends the tuple $(\mathfrak{info}, r, \phi_r)$ to the current card holder, $\mathfrak{U}_1$.

  - If $\mathfrak{U}_1$ is sending a fresh coin from his/her wallet $\mathfrak{wallet} = (s = s' + s'', t, L, \sigma_W)$ and the proof of validity holds $\mathcal{V}_r(pk_{U_2}, r, \phi_r) \stackrel{?}{=} 1$, he/she generates a proof of knowledge $\phi_\sigma \leftarrow PProveSig(pk_C, \{sk_{U_1}, s, t\}, \sigma)$. After that, $\mathfrak{U}_1$ sets $\mathfrak{info}_0 = \mathfrak{info}$, $r_0 = r$ and computes $R_0 \leftarrow \mathcal{H}(r_0, \mathfrak{info}_0)$. $\mathfrak{U}_1$ then generates the serial number $S \leftarrow \mathcal{F}_S(L, s)$ and the transference tag $T_0 \leftarrow \mathcal{F}_T(L, t, R_0)$, together with proofs of knowledge $\phi_S \leftarrow \mathcal{P}_S(L, s)$ and $\phi_{T_0} \leftarrow \mathcal{P}_T(L, t, R_0)$ of the construction, associated with the commitments in the proofs of signature ϕ_{σ_W} and $\phi_{L,l}$. Finally, $\mathfrak{U}_1$ sends the new coin $\mathfrak{coin} = (S, \phi_S, l, \phi_{L,l}, \phi_\sigma, \Pi_T = \{T_0, \phi_{T_0}, r_0, \mathfrak{info}_0\})$ to $\mathfrak{U}_2$, and keeps the updated wallet $\mathfrak{wallet}' = (s = s' + s'', t, L - 1, \sigma_W)$ if $L - 1 > 0$.
  - If $\mathfrak{U}_1$ is sending a received coin $\mathfrak{coin} = (S, \phi_S, l, \phi_{L,l}, \phi_\sigma, \Pi_T = \{T_j, \phi_{T_j}, r_j, \mathfrak{info}_j\}_{j=0\ldots h})$ and the proof of validity holds $\mathcal{V}_r(pk_{U_2}, r, \phi_r) \stackrel{?}{=} 1$, he/she first sets $\mathfrak{info}_{h+1} = \mathfrak{info}$ and $r_{h+1} = r$, and then computes $R_{h+1} \leftarrow$

$\mathcal{H}(r_{h+1}, \mathfrak{info}_{h+1})$ and $t \leftarrow \mathcal{H}(S, \{T_j\}_{j=0...h})$. $\mathfrak{U}_1$ generates a new transference tag $T_{h+1} \leftarrow \mathcal{F}_T(sk_{U_1}, t, R_{h+1})$ and a proof of knowledge $\phi_{T_{h+1}} \leftarrow \mathcal{P}_T(sk_{U_1}, t, R_{h+1})$ of the construction. Finally, $\mathfrak{U}_1$ generates the proof of ownership $\phi_{r_h} \leftarrow \mathcal{P}_r(pk_{U_1}, \mathcal{H}(\mathfrak{info}_h))$ that the coin belongs to him/her. The coin $\mathfrak{coin}' = (S, \phi_S, l, \phi_{L,l}, \phi_\sigma, \Pi_T = \{T_j, \phi_{T_j}, r_j, \mathfrak{info}_j\}_{j=0...(h+1)})$ and the proof of ownership ϕ_{r_h} are sent to $\mathfrak{U}_2$.

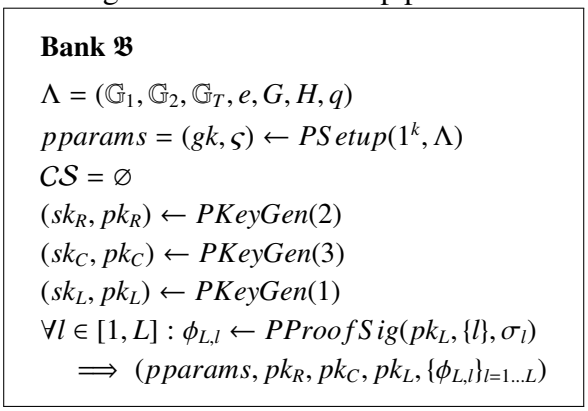
Upon reception, $\mathfrak{U}_2$ asserts if $\mathfrak{coin}'$ was a coin of $\mathfrak{U}_1$'s by $\mathcal{V}_r(pk_{U_1}, r_h, \phi_{r_h}) \stackrel{?}{=} 1$. Then, he/she verifies the construction of the serial number S by $\mathcal{V}_S(S, \phi_S) \stackrel{?}{=} 1$ and the tags $\{T_j\}_{j=0...(h+1)}$ by $\bigwedge_{j=0}^{h+1} (\mathcal{V}_T(\mathcal{H}(r_j, \mathfrak{info}_j), T_j, \phi_{T_j}) \stackrel{?}{=} 1)$, as well as if the tag of ownership r_{h+1} is the one he/she had generated (i.e., if $r \stackrel{?}{=} r_{h+1}$).

If all proofs are correct, $\mathfrak{U}_2$ stores the coin $\mathfrak{coin}'$ as his/her own.

- $Deposit\Big(\mathfrak{U}(\mathfrak{coin}) \leftrightarrow \mathfrak{B}(\mathcal{CS})\Big) \rightarrow \mathcal{DS}$: User $\mathfrak{U}$ sends to the bank $\mathfrak{B}$ a coin $\mathfrak{coin} = (S, \phi_S, l, \phi_{L,l}, \phi_\sigma, \Pi_T = \{T_j, \phi_{T_j}, r_j, \mathfrak{info}_j\}_{j=0...h})$ to be deposited. It verifies if the user had not modified the coin before by verifying if the serial number and the transference tags were correctly constructed by $\mathcal{V}_S(S, \phi_S) \stackrel{?}{=} 1 \wedge \bigwedge_{j=0}^{h} \mathcal{V}_T(\mathcal{H}(r_j, \mathfrak{info}_j), T_j, \phi_{T_j})$. $\mathfrak{B}$ simplifies the coin by removing all proofs, forming a coin $\widehat{\mathfrak{coin}} = (S, \pi_D = \{T_j, r_j, \mathfrak{info}_j\}_{j=1...h})$ that is stored in the set of deposited coins $\mathcal{CS}$. Then it verifies if there is any coin $\overline{\mathfrak{coin}}$ with identifier $\overline{S} = S$ already deposited in $\mathcal{CS}$. For each coin $\overline{\mathfrak{coin}}$, $\mathfrak{B}$ executes $Identify(\mathfrak{coin}, \overline{\mathfrak{coin}})$, retrieving the list of public keys $\mathcal{DS}$ of users who had illegally double-spent.

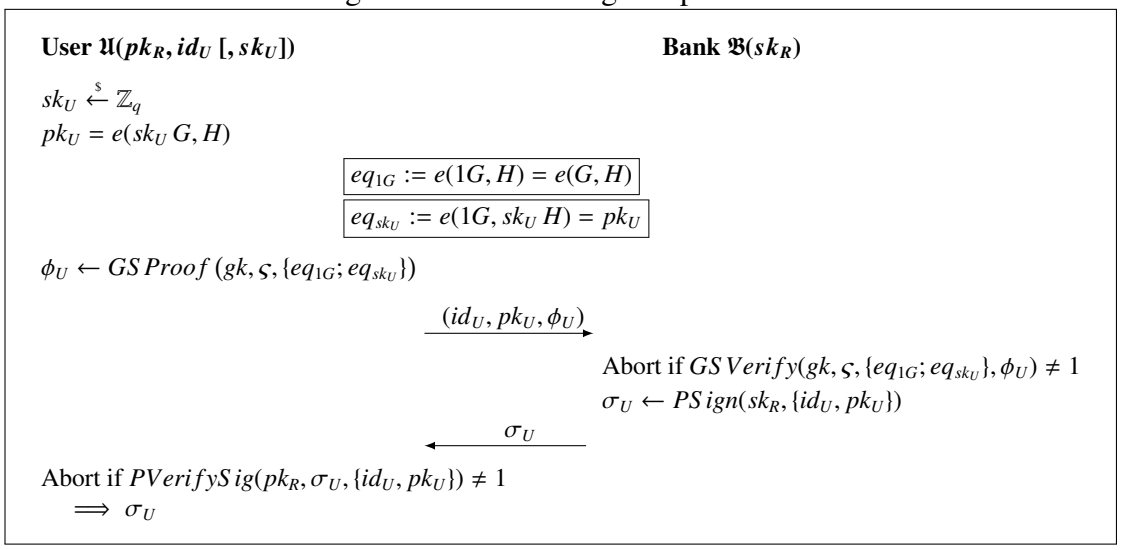
- $Identify(\mathfrak{coin}, \overline{\mathfrak{coin}}) \rightarrow pk_D$: The bank $\mathfrak{B}$ parses coins $\mathfrak{coin} = (S, \pi_D = \{T_j, r_j, \mathfrak{info}_j\}_{j=0...h})$ and $\overline{\mathfrak{coin}} = (\overline{S}, \overline{\pi_D} = \{\overline{T_j}, \overline{r_j}, \overline{\mathfrak{info}_j}\}_{j=0...\overline{h}})$ with the same serial number $S = \overline{S}$. It searches for the first index j in which $T_j \neq \overline{T_j}$, computes $R_j \leftarrow \mathcal{H}(r_j, \mathfrak{info}_j)$ and $\overline{R_j} \leftarrow \mathcal{H}(\overline{r_j}, \overline{\mathfrak{info}_j})$, and retrieves the public key of the perpetrator $\mathcal{D}$ as $pk_D = (\frac{T_j}{\overline{T_j}})^{\frac{1}{R_j - \overline{R_j}}}$.

Figure 30: E-cash: Setup protocol



Source: Author

Figure 31: E-cash: Register protocol

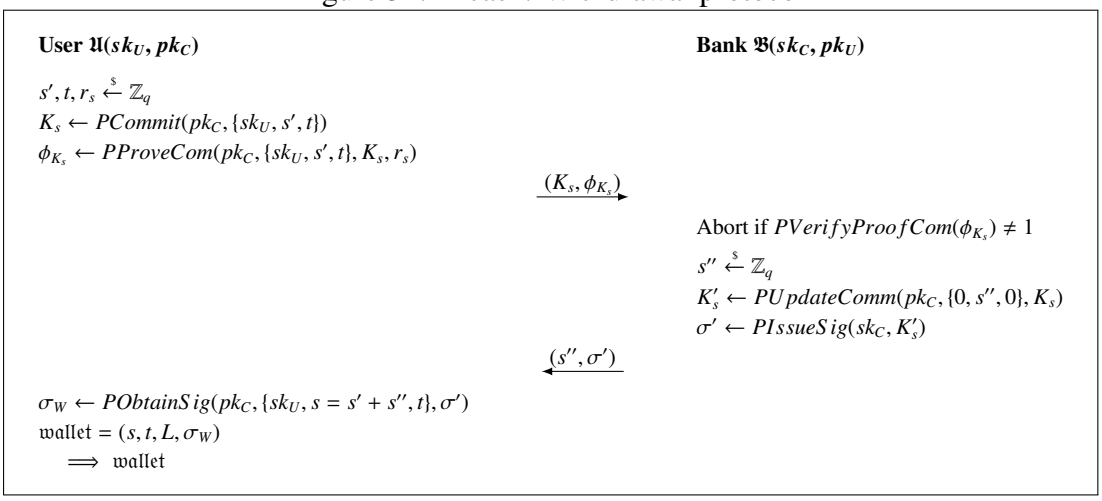


Source: Author

3.9.2 Using malleable signatures in e-cash schemes

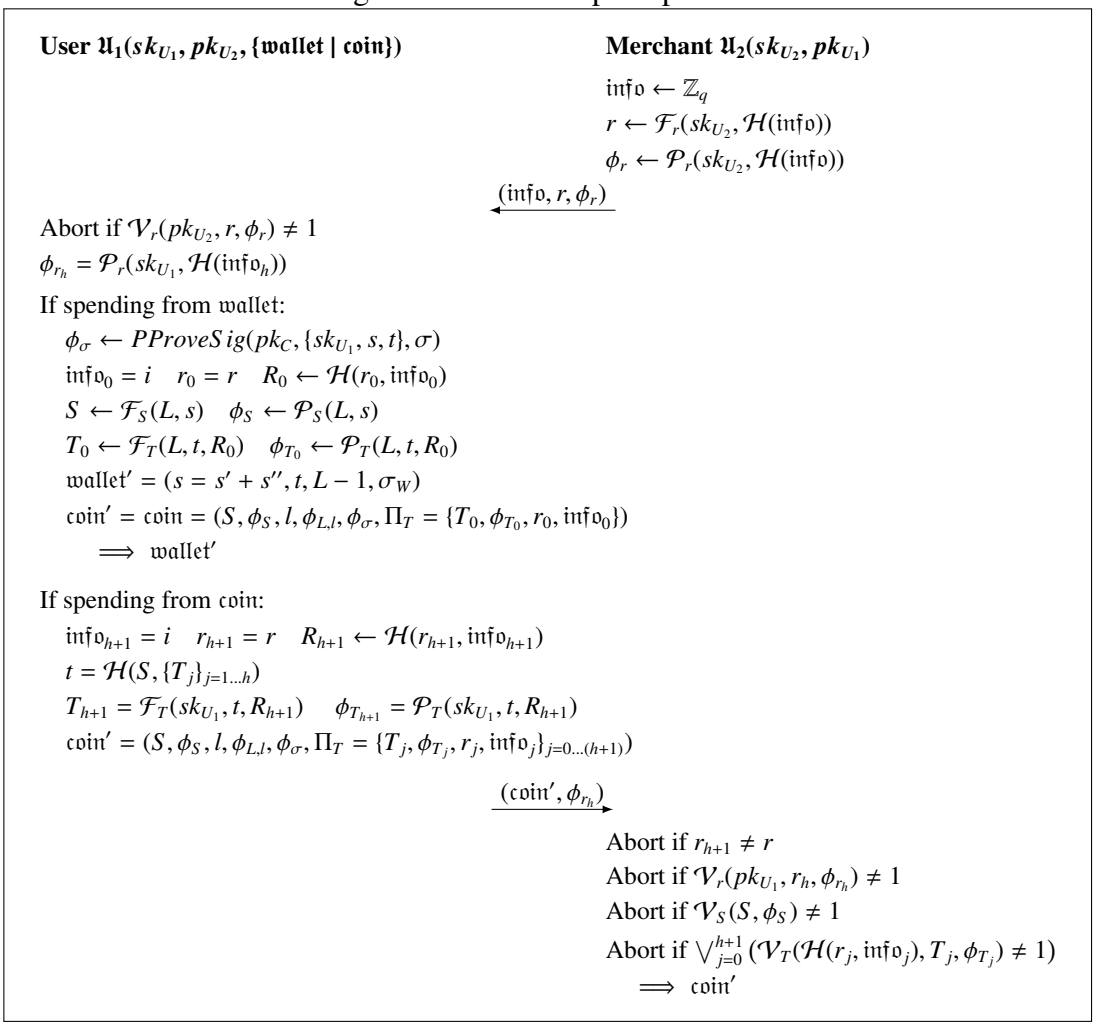
Recent works apply malleable signatures when creating transferable e-cash (BALDIMTSI et al., 2015). These signatures have the same properties of a digital signature together with a *signature evaluation* method, which transforms a known signature into another valid signature over the same signed values or over transformed values. This protocol is composed by an efficient randomizable signature scheme (e.g., (POINTCHEVAL; SANDERS, 2016) together with a non-interactive proof of knowledge. The generic construction of (CHASE et al., 2014) uses malleable proofs to create these constructions, such as Groth-Sahai proofs (CHASE et al., 2012) or succinct non-

Figure 32: E-cash: Withdrawal protocol



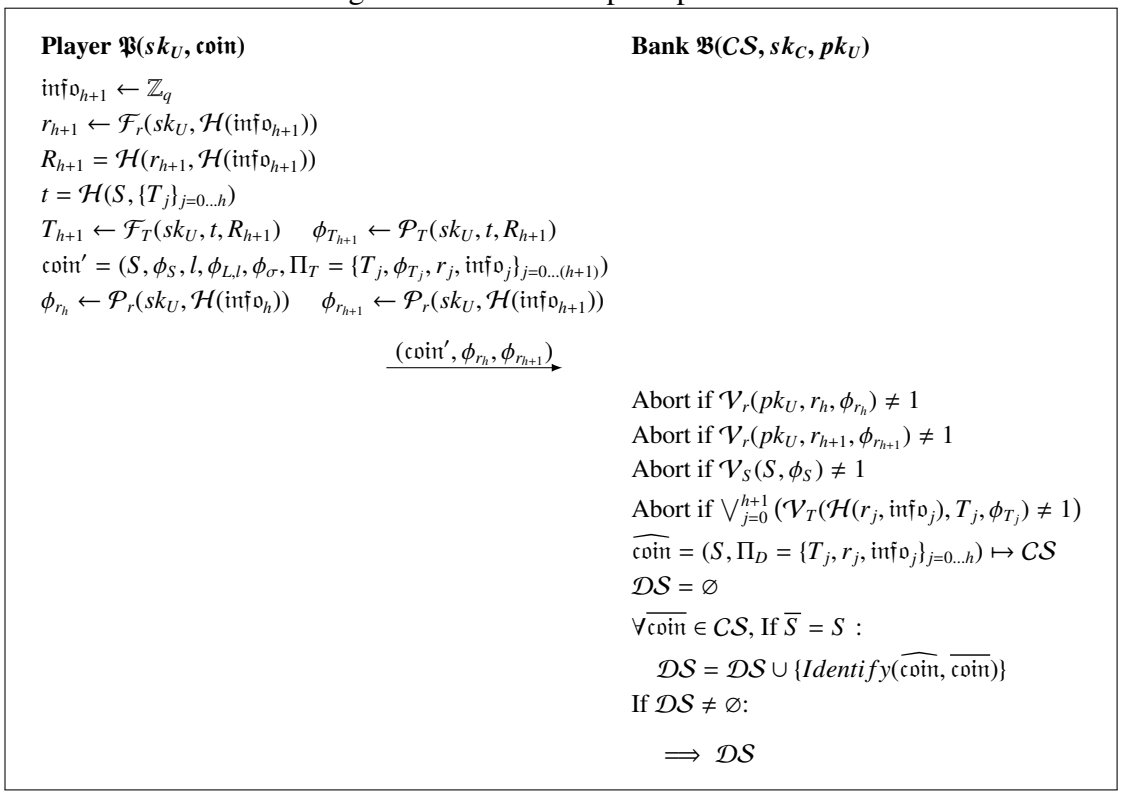
Source: Author

Figure 33: E-cash: Spend protocol



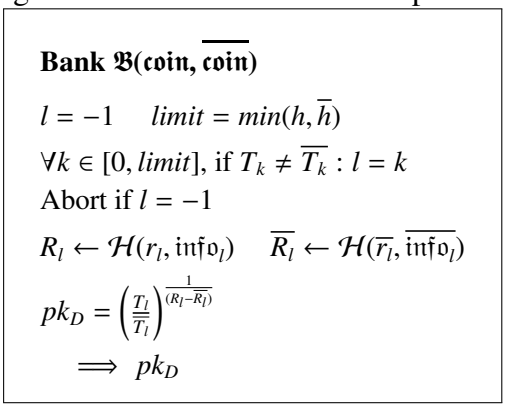
Source: Author

Figure 34: E-cash: Deposit protocol



Source: Author

Figure 35: E-cash: Identification protocol



Source: Author

interactive arguments of knowledge (SNARK) (CHASE et al., 2013).

E-cash protocols built from malleable proofs present a higher level of anonymity. Besides providing SA, it also provides PA1 and PA2, which protects the traders' privacy against the bank when it passively observes the communications and against users that owned the exchanged coins. The serial number (equivalent to S in compact e-

cash), the transference tags (equivalent to T_j), the ownership tags (equivalent to r_j), and signatures (equivalent to σ) are encrypted and randomized. Their proofs of knowledge (equivalent to ϕ_S, ϕ_{T_j}, and ϕ_σ) are randomized and mauled. As a result, the coin obtained is unlinkable to the previously owned one.

Despite the advantages of malleability, there is a larger cost involved in its operations when compared to compact e-cash protocols due to the valid transformations required for mauling coins. These transformations can be classified in two groups: the withdrawal transformation and the spending transformation.

To ensure the coin is not linkable to the withdrawn one, each spending operation needs to transform the information provided by the bank. More precisely, the signature issued by the bank when a coin is withdrawn is replaced by the one defined in (ABE et al., 2012); this leads to a larger number of point elements to the signature's representation, but reduces the number of pairing computations to its verification. Although we seek more efficiency for our protocol, we cannot replace the one from Section 3.8 because this new one is a plain signature (and not a blind signature, as required in our protocol). By applying the valid transformations for the malleable proof of knowledge, this signature can be transformed before each spending. This transformation re-randomizes the signature, and the commitments to the proof of knowledge are mauled, resulting in new point multiplications. When the signature is completely randomizable (as it is necessary for FA, PA1 and PA2), the cost is equivalent to creating an entire new proof. While it does not influence the verification execution from the receiver, the spender will have a larger cost to send this coin.

Another information from the withdrawal is the serial number, that now must be encrypted with a homomorphic encryption scheme (e.g., ElGamal (ELGAMAL, 1984)). Although the additional cost of the encryption is small (two point multiplications), the serial number must increase in size just like the transference tag, and new proofs of knowledge must be created. Each transference will have the previous serial

numbers transformed and their proofs mauled. It will result in more point multiplications to prove and additional pairing computations to verify that, as the coin grows, will continually increase.

To protect spending privacy, the information related to each transference also is transformed. Alike the serial number, the transference tag is encrypted and randomized. Each one will be randomized and its proof mauled. The resulting verification will have no additional cost, but the spender will have greater costs to transform the previous proofs and create the new one.

Providing improved anonymity will largely increase the costs for spending. The number of point multiplications will increase linearly with how many times a coin was spent, and the receiver will have additional pairing computations to verify the received coin.

3.10 Summary

In this Chapter we have introduced the mathematical and cryptographic concepts regarding finite fields, elliptic curves and bilinear pairings. Having those in mind, we defined the building blocks for a transferable e-cash scheme, focusing on the reasons for our choices and on the properties these blocks provide. Namely, we presented a proof of knowledge, a verifiable random function and provable signature, and concluded by presenting the methods for the chosen e-cash scheme (compact e-cash), its security properties and inner functions that will base the construction of our P2P TCG.

4 PROPOSED PROTOCOL

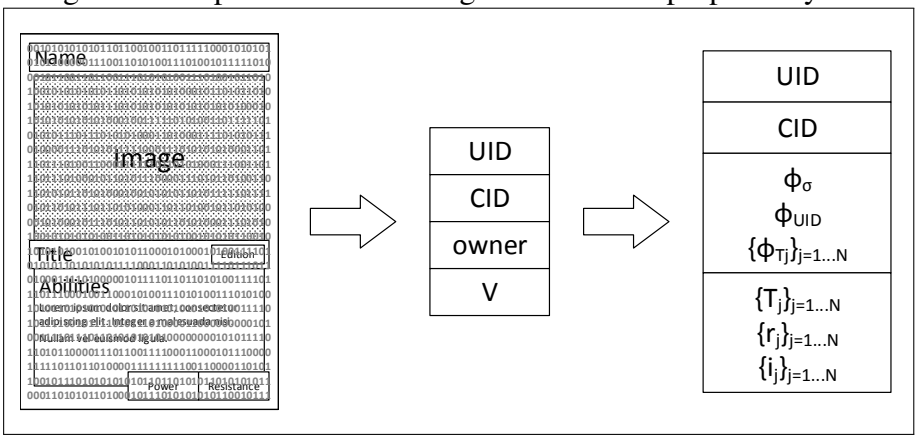
After defining the building blocks of a transferable e-cash scheme in Chapter 3, we set off for a concrete instantiation of the proposed protocol for securely trading cards. This instantiation relies on the methods and notation presented on previous chapters. We start by presenting specific notation for our construction, which comprises the definition of roles and of which values are used in the representation of our cards. Then we propose the complete protocol for securely exchanging cards.

4.1 Notation

Each method refers to roles of players $\mathfrak{P}_i$ with index (if any) i, and of the game server $\mathfrak{G}$, comprised by the registration center $\mathfrak{C}$, card market $\mathfrak{M}$ and game auditor $\mathfrak{A}$.

A card $\mathfrak{card}$ is represented by the tuple $\mathfrak{card} = (UID, CID, V, owner)$, where: $UID \in \mathbb{G}_1$ is its unique identifier; $CID \in \mathbb{Z}_q$ is the numeric representation of the card's class using some suitable encoding; $V = (\phi_{UID}, \phi_\sigma, \{\phi_{T_j}\}_{j=1...N})$, where ϕ_{UID}, ϕ_σ and ϕ_{T_j} are, respectively, proofs of knowledge of the construction of the UID, of the signature from the market, and of the j-th transference tag; and $owner = \Pi_T = \{T_j, r_j, \mathfrak{info}_j\}_{j=1...N}$ corresponds to the records of all owners of the cards, so that, for each index $j \in [1, N]$, T_j is the transference tag, r_j is the ownership tag and $\mathfrak{info}_j$ is the public information regarding the transference. Figure 36 exemplify this representation.

Figure 36: Representation of a digital card in the proposed system



Source: Author

4.2 Construction

The operations comprised by the proposed scheme are, then:

Setup

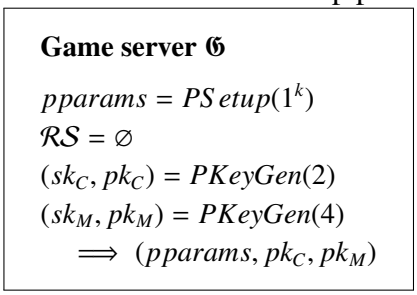
This method generates the public parameters of the system, enclosing the underlying pairing setting, proof of knowledge, the verifiable random function and the signature scheme. It also generates the key pair for registering new players and for stamping new cards. Figure 37 summarizes the operations.

$Setup(1^k) \rightarrow (pparams, pk_C, pk_M)$

The game server generates the system parameters, that is the same of a signature setup $pparams = (gk, \varsigma) \leftarrow PSetup(1^k)$, which comprises the asymmetric pairing setting $\Lambda = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H, q)$ in the setup of a Groth-Sahai proof (gk, ς). These parameters are used by the subsequent operations and, for shortness, are omitted in their descriptions.

It initializes an empty list of reported cards $\mathcal{RS} = \varnothing$. The game server also generates two key-pairs: $(sk_C, pk_C) \leftarrow PKeyGen(2)$ to register players and $(sk_M, pk_M) \leftarrow PKeyGen(4)$ to stamp cards. It then publishes $(pparams, pk_C, pk_M)$.

Figure 37: SecureTrade: Setup protocol



Source: Author

Register

This method allows a user to enroll in the game server, allowing him to be verified as a valid player in the system. It also allows the game server to receive the public key associated to the player's ID, so that he/she can be identified if he/she cheats. Figure 38 summarizes the operations.

$$Register\Big(\mathfrak{P}(pk_C, id_P[, sk_P]) \leftrightarrow \mathfrak{C}(sk_C)\Big) \rightarrow \sigma_P$$

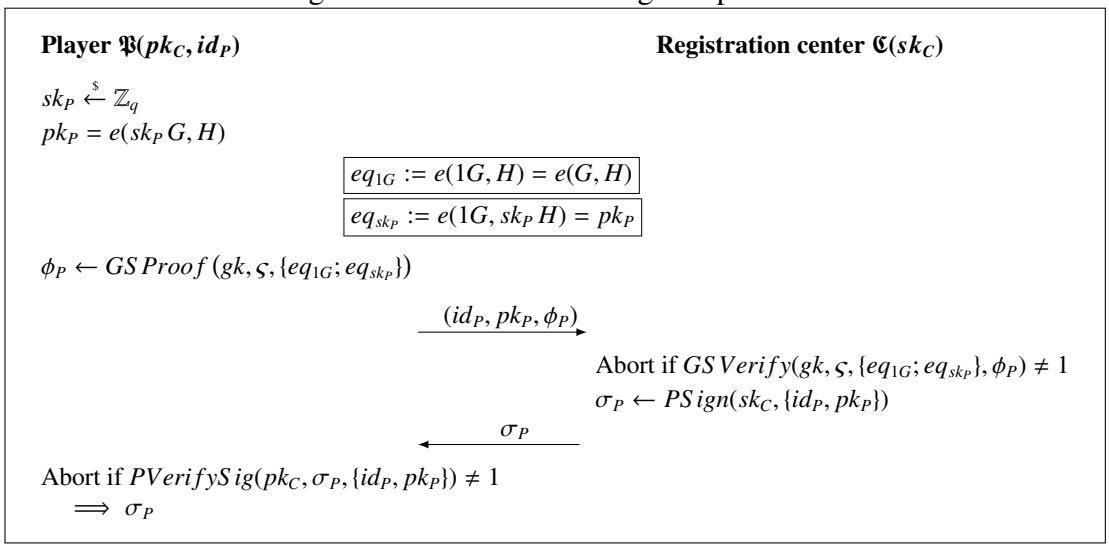
Player $\mathfrak{P}$ with identity id_P generates a secret key $sk_P \xleftarrow{\$} \mathbb{Z}_q$ and computes the public key $pk_P = e(G, sk_P H)$. $\mathfrak{P}$ generates a proof of knowledge $\phi_P = (\vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1,2}) \leftarrow GSProof\,(gk, \varsigma, \{e(1G, sk_P H) = pk_P; e(1G, H) = e(G, H)\}, \{1G\}, \{sk_P H\})$. The triple (id_P, pk_P, ϕ_P) is sent to the registration center $\mathfrak{C}$.

If the proof ϕ_P is valid by $GSVerify(gk, \varsigma, \{e(1G, sk_P H) = pk_P; e(1G, H) = e(G, H)\}, \vec{C}, \vec{D}, \{\pi_k, \varphi_k\}_{k=1,2}) \stackrel{?}{=} 1$, $\mathfrak{C}$ generates a signature $\sigma_P \leftarrow PSign(sk_C, \{id_P, pk_P\})$ and sends it to $\mathfrak{P}$. If the verification $PVerifySig(pk_C, \sigma_P, \{id_P, pk_P\}) \stackrel{?}{=} 1$ is accepted, $\mathfrak{P}$ can then present σ_P as his/her certificate.

Stamp

This method is used so that a registered player can stamp a new valid card. The card retrieved by this method is unknown to the card market, so that it is unlinkable to the instance of the stamping. Figure 39 summarizes the operations.

Figure 38: SecureTrade: Register protocol



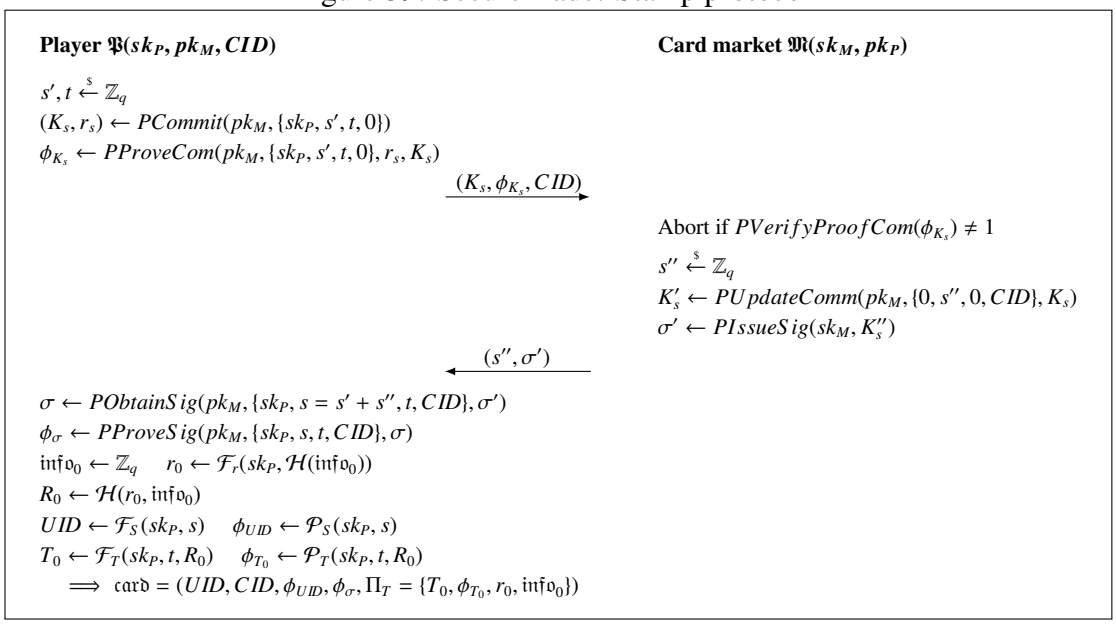
Source: Author

$$Stamp\Big(\mathfrak{P}(sk_P, pk_M, CID) \leftrightarrow \mathfrak{M}(sk_M, pk_M)\Big) \rightarrow \mathfrak{card}:$$

To purchase an instance of a card with class CID, player $\mathfrak{P}$ generates a partial identifier seed $s' \xleftarrow{\$} \mathbb{Z}_q$ and a transference seed $t \xleftarrow{\$} \mathbb{Z}_q$, and the card market $\mathfrak{M}$ generates the card's partial identifier component $s'' \xleftarrow{\$} \mathbb{Z}_q$. Both parties execute the interactive protocol to obtain a blind signature $\sigma \leftarrow \Big(PObtainSig(pk_M, \{sk_P, s', t, 0\}) \leftrightarrow PIssueSig(sk_M, \{0, s'', 0, CID\})\Big)$ that is returned to $\mathfrak{P}$ together with s''.

The player then generates a proof of knowledge $\phi_\sigma \leftarrow PProveSig(pk_M, \{sk_P, s = s' + s'', t, CID\}, \sigma)$. After that, $\mathfrak{P}$ chooses some unique public information $\mathfrak{info}_0 \leftarrow \{0,1\}^*$ (e.g., a timestamp) and computes $r_0 \leftarrow \mathcal{F}_r(sk_P, \mathcal{H}(\mathfrak{info}_0))$ and $R_0 \leftarrow \mathcal{H}(r_0, \mathfrak{info}_0)$. $\mathfrak{P}$ then generates the unique identifier $UID \leftarrow \mathcal{F}_S(sk_P, s)$ and the transference tag $T_0 \leftarrow \mathcal{F}_T(sk_P, t, R_0)$, together with proofs of knowledge $\phi_{UID} \leftarrow \mathcal{P}_S(sk_P, s)$ and $\phi_{T_0} \leftarrow \mathcal{P}_T(sk_P, t, R_0)$ of the construction, associated with the commitments in the proof of signature ϕ_σ. Finally, the player stores the card $\mathfrak{card} = (UID, CID, \phi_{UID}, \phi_\sigma, \Pi_T = \{T_0, \phi_{T_0}, r_0, \mathfrak{info}_0\})$.

Figure 39: SecureTrade: Stamp protocol



Source: Author

Send

This method allows a player (the sender) to send a card to another one (the receiver). If the sender continues using the card, the game server can identify the transgressor when the usage is reported by an opponent of his/hers. Besides, the receiver becomes the new owner of the card and will be the only one able to prove ownership of the card. Figure 40 summarizes the operations.

$Send\Big(\mathfrak{P}_1(sk_{P_1}, pk_{P_2}, \mathfrak{card}) \leftrightarrow \mathfrak{P}_2(sk_{P_2}, pk_{P_1})\Big) \rightarrow \mathfrak{card}'$:

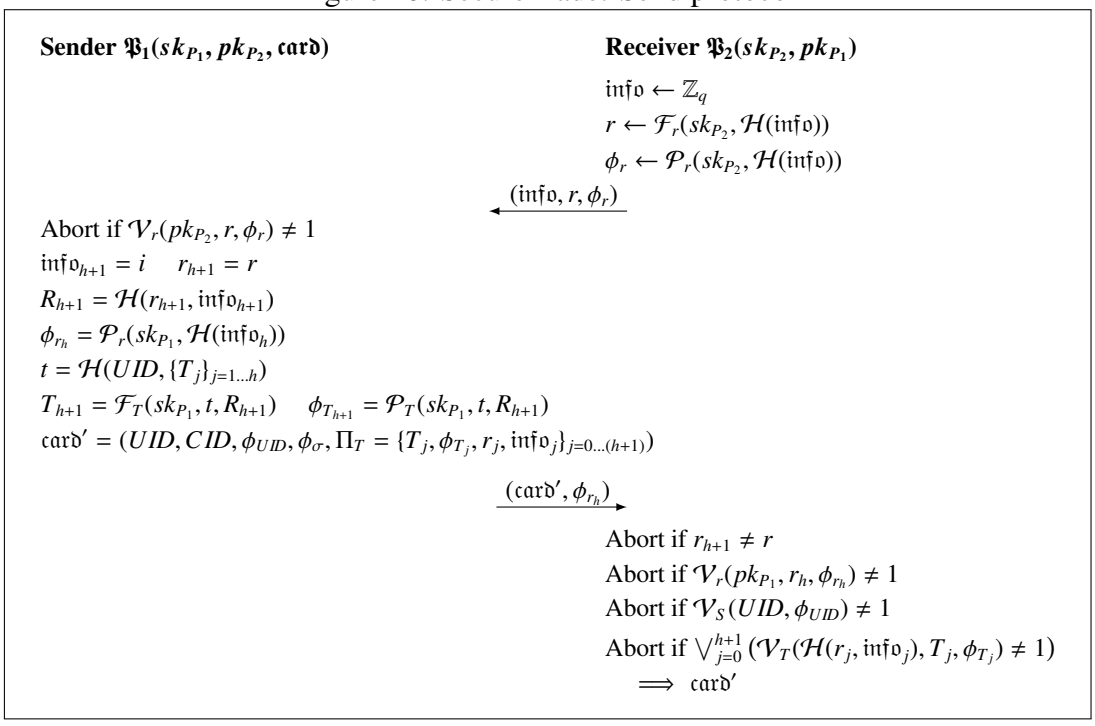
The receiver $\mathfrak{P}_2$ chooses some public information $\mathfrak{info} \leftarrow \{0,1\}^*$ (e.g., a timestamp) and computes $r \leftarrow \mathcal{F}_r(sk_{P_2}, \mathcal{H}(\mathfrak{info}))$ and a proof of validity $\phi_r \leftarrow \mathcal{P}_r(sk_{P_2}, \mathcal{H}(\mathfrak{info}))$. $\mathfrak{P}_2$ then sends the tuple $(\mathfrak{info}, r, \phi_r)$ to the current card holder, $\mathfrak{P}_1$.

$\mathfrak{P}_1$ parses $\mathfrak{card} = (UID, CID, \phi_{UID}, \phi_\sigma, \Pi_T = \{T_j, \phi_{T_j}, r_j, \mathfrak{info}_j\}_{j=0\ldots h})$ and verifies the proof of validity $\mathcal{V}_r(pk_{P_2}, r, \phi_r) \stackrel{?}{=} 1$. If everything is correct, $\mathfrak{P}_1$ first sets $\mathfrak{info}_{h+1} = \mathfrak{info}$ and $r_{h+1} = r$, and then computes $R_{h+1} \leftarrow \mathcal{H}(r_{h+1}, \mathfrak{info}_{h+1})$ and $t \leftarrow \mathcal{H}(UID, \{T_j\}_{j=0\ldots h})$. $\mathfrak{P}_1$ generates a new transference tag $T_{h+1} \leftarrow \mathcal{F}_T(sk_{P_1}, t, R_{h+1})$ and a proof of knowledge $\phi_{T_{h+1}} \leftarrow \mathcal{P}_T(sk_{P_1}, t, R_{h+1})$ of the construction. Finally, $\mathfrak{P}_1$

generates the proof of ownership $\phi_{r_h} \leftarrow \mathcal{P}_r(pk_{P_1}, \mathcal{H}(\mathfrak{info}_h))$ that the card belongs to him/her. The card $\mathfrak{card}' = (UID, CID, \phi_{UID}, \phi_\sigma, \Pi_T = \{T_j, \phi_{T_j}, r_j, \mathfrak{info}_j\}_{j=0\ldots(h+1)})$ and the proof of ownership ϕ_{r_h} are sent to $\mathfrak{P}_2$.

Upon reception, $\mathfrak{P}_2$ asserts if $\mathfrak{card}'$ was a card of $\mathfrak{P}_1$'s by $\mathcal{V}_r(pk_{P_1}, r_h, \phi_{r_h}) \stackrel{?}{=} 1$. Then, he/she verifies the construction of the unique identifier UID by $\mathcal{V}_S(UID, \phi_{UID}) \stackrel{?}{=} 1$ and the tags $\{T_j\}_{j=0\ldots(h+1)}$ by $\bigwedge_{j=0}^{h+1}(\mathcal{V}_T(\mathcal{H}(r_j, \mathfrak{info}_j), T_j, \phi_{T_j}) \stackrel{?}{=} 1)$, as well as if the tag of ownership r_{h+1} is the one he/she had generated (i.e., if $r \stackrel{?}{=} r_{h+1}$). If all proofs are correct, $\mathfrak{P}_2$ stores the card $\mathfrak{card}'$ as his/her own.

Figure 40: SecureTrade: Send protocol



Source: Author

Play

This method allows a player to prepare his/her cards so that they can be used in a match with other players (the opponents). The used card still belongs to the first player, and no opponent is able to send or play with the card even after the owner has applied this method. Any opponent may deny the match if the used card is not a valid

one. Figure 41 summarizes the operations.

$Play\Big(\mathfrak{P}_1(sk_{P_1}, \mathfrak{card}) \leftrightarrow \mathfrak{P}_2(pk_{P_1})\Big)$: Player $\mathfrak{P}_1$ prepares a card $\mathfrak{card} = (UID, CID, \phi_{UID}, \phi_\sigma, \Pi_T = \{T_j, \phi_{T_j}, r_j, \mathfrak{info}_j\}_{j=0\ldots h})$ that has been updated h times. $\mathfrak{P}_1$ chooses some public information $\mathfrak{info}_{h+1} \leftarrow \{0,1\}^*$ and computes $r_{h+1} \leftarrow \mathcal{F}_r(sk_{P_1}, \mathcal{H}(\mathfrak{info}_{h+1}))$, $R_{h+1} \leftarrow \mathcal{H}(r_{h+1}, \mathfrak{info}_{h+1})$ and $t \leftarrow \mathcal{H}(UID, \{T_j\}_{j=0\ldots h})$. Then $\mathfrak{P}_1$ generates a new transference tag $T_{h+1} \leftarrow \mathcal{F}_T(sk_{P_1}, t, R_{h+1})$, together with proof of knowledge $\phi_{T_{h+1}} \leftarrow \mathcal{P}_T(sk_{P_1}, t, R_{h+1})$ of the construction. The card $\mathfrak{card}$ is updated to $\mathfrak{card}' = (UID, CID, \phi_{UID}, \phi_\sigma, \Pi_T = \{T_j, \phi_{T_j}, r_j, \mathfrak{info}_j\}_{j=0\ldots(h+1)})$. $\mathfrak{P}_1$ also prepares two proofs of ownership $\phi_{r_h} \leftarrow \mathcal{P}_r(sk_{P_1}, \mathcal{H}(\mathfrak{info}_h))$ and $\phi_{r_{h+1}} \leftarrow \mathcal{P}(sk_{P_1}, \mathcal{H}(\mathfrak{info}_{h+1}))$ to prove that the card was correctly prepared. The triple $(\mathfrak{card}', \phi_{r_h}, \phi_{r_{h+1}})$ is sent to his/her opponent $\mathfrak{P}_2$ (that can actually be several players).

Upon reception, $\mathfrak{P}_2$ asserts if $\mathfrak{card}'$ belongs to $\mathfrak{P}_1$ by $\mathcal{V}_r(pk_{P_1}, r_h, \phi_{r_h}) \stackrel{?}{=} 1 \wedge \mathcal{V}_r(pk_{P_1}, r_{h+1}, \phi_{r_{h+1}}) \stackrel{?}{=} 1$. Then, he/she verifies the construction of the unique identifier UID by $\mathcal{V}_S(UID, \phi_{UID})$ and transference tags $\{T_j\}_{j=0\ldots(h+1)}$ by $\bigwedge_{j=0}^{h+1}(\mathcal{V}_T(\mathcal{H}(r_j, \mathfrak{info}_j), T_j, \phi_{T_j}) \stackrel{?}{=} 1)$. If they are all valid, $\mathfrak{P}_2$ then stores this card $\mathfrak{card}'$ locally, so it can report this information to the game server later, and uses the unique identifier UID to identify this card during the match.

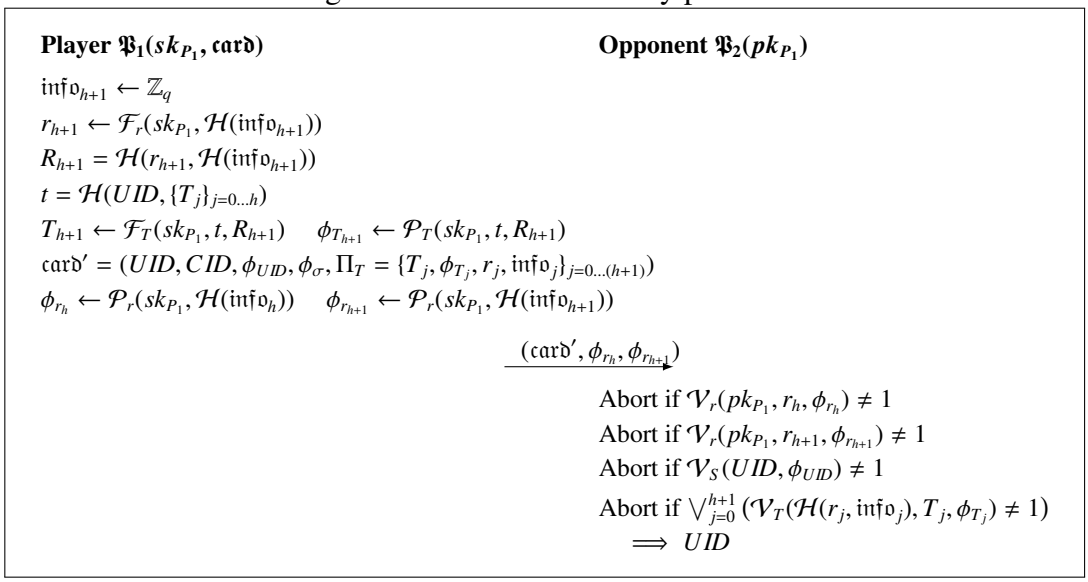
Report

This method allows any player to submit a report of valid cards that their opponents had played with, altogether with any other information relevant to the system (e.g., the match result, in-game cheating proof). The reported cards are stored in a list with only relevant information to identify any player that had illegally duplicated cards. Figure 42 summarizes the operations.

$Report\Big(\mathfrak{P}(\mathfrak{card}) \leftrightarrow \mathfrak{A}(\mathcal{RS})\Big) \rightarrow \mathcal{DS}$:

Player $\mathfrak{P}$ sends to the game auditor $\mathfrak{A}$ a card $\mathfrak{card} = (UID, CID, \phi_{UID}, \phi_\sigma, \Pi_T =$

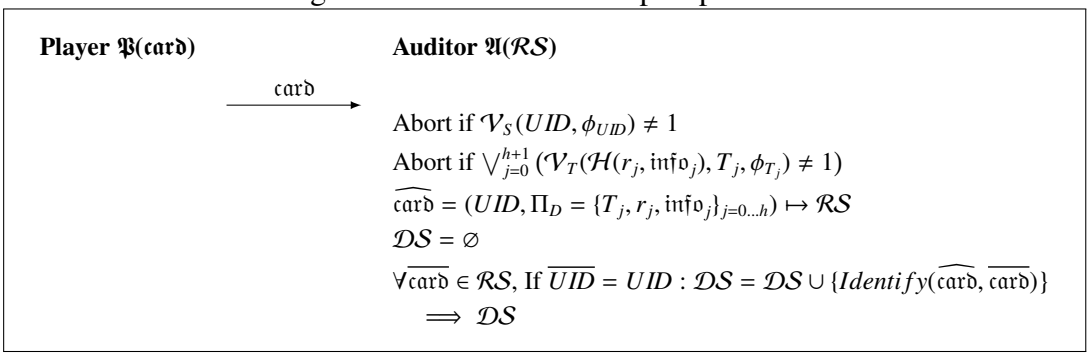
Figure 41: SecureTrade: Play protocol



Source: Author

$\{T_j, \phi_{T_j}, r_j, \mathfrak{info}_j\}_{j=0...h}$) that an opponent has used in some match. It verifies if the player had not modified the card before he/she reported by verifying if the unique identifier and the transference tags were correctly constructed by $\mathcal{V}_S(UID, \phi_{UID}) \stackrel{?}{=} 1 \wedge \bigwedge_{j=0}^{h} \mathcal{V}_T(\mathcal{H}(r_j, \mathfrak{info}_j), T_j, \phi_{T_j})$. $\mathfrak{A}$ simplifies the card by removing all proofs, forming a card $\widehat{\mathfrak{card}} = (UID, \Pi_D = \{T_j, r_j, \mathfrak{info}_j\}_{j=1...h})$ that is stored in the set of reported cards $\mathcal{RS}$. Then it verifies if there is any card $\overline{\mathfrak{card}}$ with identifier $\overline{UID} = UID$ already reported in $\mathcal{RS}$. For each card $\overline{\mathfrak{card}}$, $\mathfrak{A}$ executes $Identify(\widehat{\mathfrak{card}}, \overline{\mathfrak{card}})$, retrieving the list of public keys $\mathcal{DS}$ of users who had illegally duplicated this card.

Figure 42: SecureTrade: Report protocol



Source: Author

Refresh

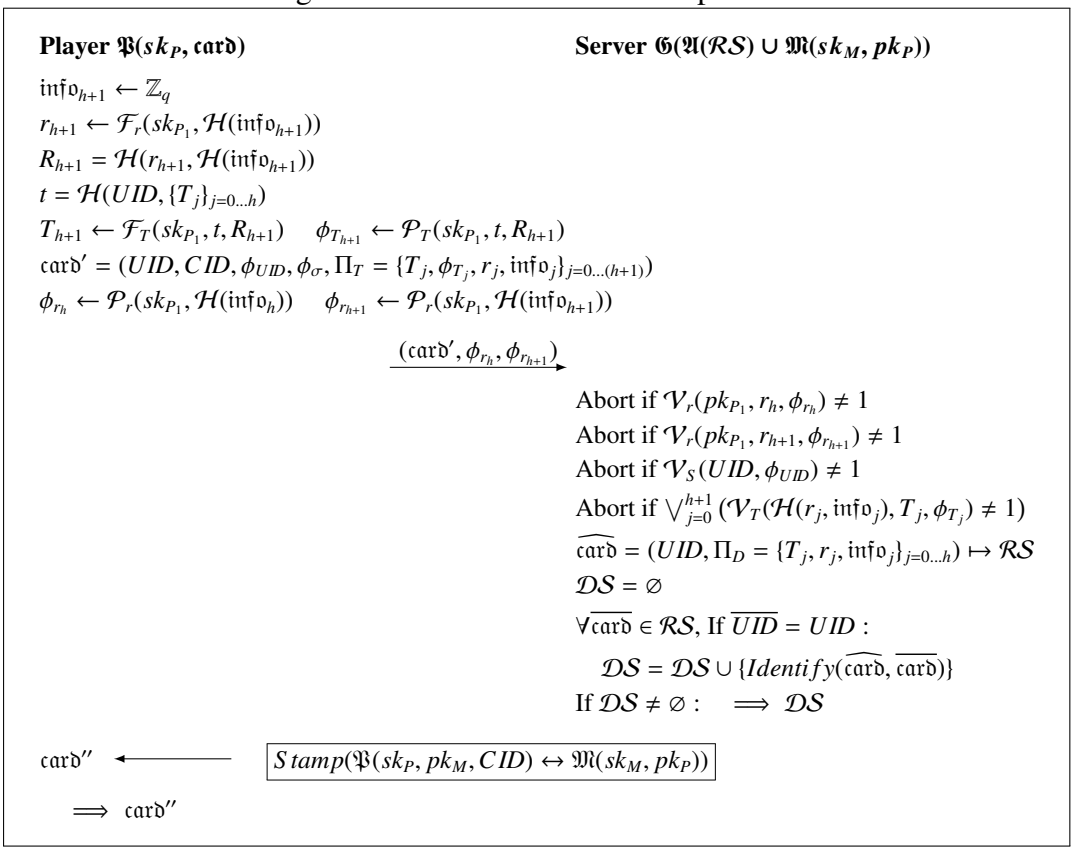
This method is used to lessen the size of some stamped card. First the game server verifies that the card is valid, then that the card belongs to the player who wishes to refresh it. Afterwards, it stores the bigger card in the list of reported cards to avoid duplication by the last owner, and stamps a mint version of a card with the same class. Figure 43 summarizes the operations.

$$Refresh\Big(\mathfrak{P}(sk_P, pk_M, \mathfrak{card}) \leftrightarrow (\mathfrak{G} = \mathfrak{A}(\mathcal{RS}) \cup \mathfrak{M}(sk_M, pk_P))\Big) \rightarrow (\mathfrak{card}''|\mathcal{DS}):$$

Player $\mathfrak{P}$ prepares a card $\mathfrak{card} = (UID, CID, \phi_{UID}, \phi_\sigma, \Pi_T = \{T_j, \phi_{T_j}, r_j, \mathfrak{info}_j\}_{j=0...h})$ that has been updated h times. $\mathfrak{P}$ chooses some public information $\mathfrak{info}_{h+1} \leftarrow \{0,1\}^*$ (e.g., a timestamp) and computes $r_{h+1} \leftarrow \mathcal{F}_r(sk_P, \mathcal{H}(\mathfrak{info}_{h+1}))$, $R_{h+1} \leftarrow \mathcal{H}(r_{h+1}, \mathfrak{info}_{h+1})$ and $t \leftarrow \mathcal{H}(UID, \{T_j\}_{j=0...h})$. Then $\mathfrak{P}$ generates a new transference tag $T_{h+1} \leftarrow \mathcal{F}_T(sk_P, t, R_{h+1})$, together with proof of knowledge $\phi_{T_{h+1}} \leftarrow \mathcal{P}_T(sk_P, t, R_{h+1})$ of the construction. The card $\mathfrak{card}$ is updated to $\mathfrak{card}' = (UID, CID, \phi_{UID}, \phi_\sigma, \Pi_T = \{T_j, \phi_{T_j}, r_j, \mathfrak{info}_j\}_{j=0...(h+1)})$. At last, $\mathfrak{P}$ produces two proofs of ownership $\phi_{r_h} = \mathcal{P}(sk_P, \mathcal{H}(\mathfrak{info}_h))$ and $\phi_{r_{h+1}} = \mathcal{P}(sk_P, \mathcal{H}(\mathfrak{info}_{h+1}))$, that $\mathfrak{card}'$ belongs to him/her, and sends the tuple $(\mathfrak{card}', \phi_{r_h}, \phi_{r_{h+1}})$ to the game server $\mathfrak{G}$.

The game auditor $\mathfrak{A}$ verifies the validity of the card by verifying if the unique identifier and the transference tags were correctly constructed by $\mathcal{V}_S(UID, \phi_{UID}) \stackrel{?}{=} 1 \wedge \bigwedge_{j=0}^{h} \mathcal{V}_T(\mathcal{H}(r_j, \mathfrak{info}_j), T_j, \phi_{T_j})$, and verifies if the card really belongs to $\mathfrak{P}$ by verifying $\mathcal{V}_r(pk_P, r_h, \phi_{r_h}) \stackrel{?}{=} 1 \wedge \mathcal{V}_r(pk_P, r_{h+1}, \phi_{r_{h+1}}) \stackrel{?}{=} 1$. If they are valid, $\mathfrak{A}$ simplifies the card by removing all proofs, forming a card $\widehat{\mathfrak{card}} = (UID, \Pi_D = \{T_j, r_j, \mathfrak{info}_j\}_{j=1...h})$ that is stored in the set of reported cards $\mathcal{RS}$. Then if verifies if there is any card $\overline{\mathfrak{card}}$ with identifier $\overline{UID} = UID$ already reported in $\mathcal{RS}$. For each card $\overline{\mathfrak{card}}$, $\mathfrak{A}$ executes $Identify(\mathfrak{card}', \overline{\mathfrak{card}})$, retrieving the list of public keys $\mathcal{DS}$ of users who had illegally duplicated this card. If identifying $\mathfrak{card}'$ did not return any transgressor, both parties execute $Stamp(\mathfrak{P}(sk_P, pk_M, CID) \leftrightarrow \mathfrak{M}(sk_M, pk_P))$ to produce a fresh card $\mathfrak{card}''$ to $\mathfrak{P}$.

Figure 43: SecureTrade: Refresh protocol



Source: Author

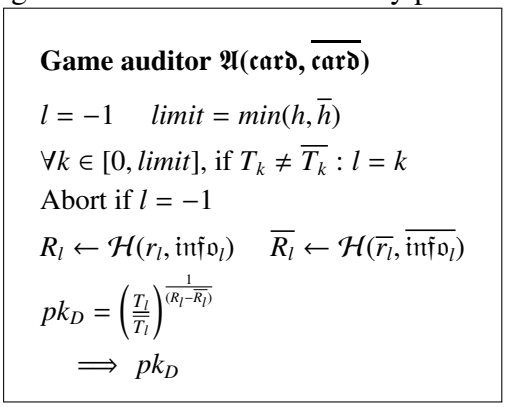
Identify

This method identifies the perpetrator that had duplicated the reported/refreshed card by comparing information related to previous and current owners of two cards with the same unique identifier. Figure 44 summarizes the operations.

$Identify(\mathfrak{card}, \overline{\mathfrak{card}}) \rightarrow pk_D$:

The game auditor $\mathfrak{A}$ parses cards $\mathfrak{card} = (UID, \Pi_D = \{T_j, r_j, \mathfrak{info}_j\}_{j=0...h})$ and $\overline{\mathfrak{card}} = (\overline{UID}, \overline{\Pi_D} = \{\overline{T_j}, \overline{r_j}, \overline{\mathfrak{info}_j}\}_{j=0...\overline{h}})$ with the same identifier $UID = \overline{UID}$. It searches for the first index l in which $T_l \neq \overline{T_l}$, computes $R_l \leftarrow \mathcal{H}(r_l, \mathfrak{info}_l)$ and $\overline{R_l} \leftarrow \mathcal{H}(\overline{r_l}, \overline{\mathfrak{info}_l})$, and retrieves the public key of the perpetrator $\mathcal{D}$ as $pk_D = (\frac{T_l}{\overline{T_l}})^{\frac{1}{R_l - \overline{R_l}}}$. If the index l is larger than the number of hops for any card (h or $\overline{h}$), this card had already been reported but had not been duplicated, so the output is empty.

Figure 44: SecureTrade: Identify protocol



Source: Author

4.3 Summary

In this section we have described an e-cash based protocol for securely trading cards in a TTP-free TCG. Our solution, based on (BELENKIY et al., 2009), applies methods *Setup*, *Register*, *Stamp*, *Send*, *Play*, *Report*, *Refresh* and *Identify*. For each method, we detailedly present all operation required to execute it by using methods described in Section 3. From now, we can analyze the efficiency of our proposed solution.

5 ANALYSIS

In this chapter, we analyze qualitatively and quantitatively the protocol described in Chapter 4 in different levels. First, we analyze the compliance with the security requirements we have defined in Section 2.3 for a P2P TCG. This description opens the discussion of what should the game server do when it encounters an illegal duplication. Then, we analyze the efficiency of the protocol using a known library for cryptographic applications. We complete the analysis with the discussion on methods to reduce load on less powerful devices.

5.1 Security analysis

In this section, we analyze the security requirements of a secure card trading system and discuss how they are fulfilled by the underlying e-cash scheme, identifying the underlying blocks that provide each property.

5.1.1 Verifiable stamping

The use of digital signatures and of proofs of knowledge ensures that stamped cards can be verified without new contacts with the game server.

By applying the P-signature scheme from Section 3.8, the first owner can verify that the card was correctly signed by the card market and complete the production of the first valid hop. To forward this information to receivers, the card owner creates non-interactive proofs of knowledge, as described in Section 3.6, thus guaranteeing the

correctness of the relayed information. For the scheme to work correctly, the market's public key must be published to all players (e.g., when they register into the system).

5.1.2 TTP-free transferability

Transferability is obtained by creating proofs of knowledge: then anyone can verify that all values created by previous owners are correct, including the values from stamping (unique identifier and market signature) and the values for each transference (transference and ownership tags). The construction of non-interactive proofs of knowledge allows subsequent owners to verify the correctness of a card that was transferred among several owners without contacting them. It is important to note that transferability only proves correctness, but does not prevent illegal duplication (identified by the *cheating detection* process) or prevent the creation of forged cards (which can be identified as fake by *verifiable stamping*).

5.1.3 Anonymity

Weak Anonymity is attained by attaching the proof of knowledge to a blind signature.

Blind signatures are commonly found in e-cash schemes to protect the identity of a user when spending cash. Whereas the bank cannot link a withdrawal to a spending in the e-cash context, the game server cannot link the stamping (variant of a withdrawal) to the usage or trading of a card (both methods derived from the underlying e-cash's algorithm spending process). Since a fresh card is obtained using the same method employed for creating a new stamped card, the equivalence is maintained.

When coupled to the transferability property, the trades can be linked to each other. To protect against passive adversaries, zero-knowledge proofs of knowledge hide the outputs from VRFs and the blind signature, which guarantees that they cannot be linked. Thus, an adversary that observes the data that represents the card cannot

link it to the current or previous owners. However, an adversary that receives the card, or that observes the trade, is able to recognize the card because the unique identifier is not modified when the card is traded. This problem can be solved in a straightforward manner, though, as it suffices for the players to communicate using an authenticated and encrypted channel, such as Bluetooth® Security Mode 2, 3 or 4 (NIST, 2012b). Since this is a standard security procedure, the encryption of the cards' identifiers are not directly treated by the proposed protocol.

5.1.4 Balance

There are two methods an adversary can use to create more cards than the market has stamped: it can either forge a new card or duplicate a valid card.

The adoption of digital signatures prevents forgery attempts by entities that do not know the market's private key, although if an adversary gains control over the market itself, it can create as many cards as desired. Honest players should not accept cards released by any entity that is not the market, i.e., that are not correctly signed by it. Additionally, if some transgressor has illegally duplicated a valid card, already signed by the card market, the TCG server can break the anonymity of this user by the identifying his/her public key. Thereby, both the signature scheme and the identification method from the e-cash scheme are the bases to achieve this property.

5.1.5 Cheating detection

The cheating detection mechanism is directly deployed by the identification protocol from the e-cash scheme. The process of identifying double-spenders is usually done by crossing information from the two instances of the duplicated cards, either from reports or refreshed cards. Following the enumeration of cheating types discussed in Section 2.2, the detection occurs as follows:

- *Double-refresh*: If the market receives the same card to be refreshed twice, it can verify that the card with the same universal identifier UID and having the same number of transferences (i.e., the same length of Π_T) already has a fresh version and will reject the second refresh.

- *Double-trade*: When two instances of a same card are reported or refreshed, the game auditor can identify the transgressor by the first index in which each the instances have different owners (i.e., the smallest j such that T_j differs). It can then use these instances as proof for identifying the transgressor.

- *Trade-then-play*: When a valid instance of a card is reported or refreshed, and the subsequent match is reported, the game auditor can identify the transgressor by the index of the match (the last T from the reported card). The reported card and the valid instance can then be used as proof that the player has cheated.

- *Refresh-then-trade*: When a card that was refreshed is sent to another player, and then reported or refreshed, the game auditor can identify the transgressor by the index of the refreshing (the last T from the refreshed card). The game auditor can then use the refreshed card and the traded one as proof of the transgression.

- *Refresh-then-play*: When a card that was refreshed is used in a match and then reported, the game auditor can identify the transgressor by the index of the match and the card refreshed (the last T from the refreshed card). The refreshed card and the reported one can be used as proof that the cheating occurred.

5.1.6 Exculpability

Exculpability is also dependent on the identification of the illegal duplicator, similarly to what occurs in e-cash schemes.

The equation to compute the public key of the transgressor depends on the transference tags (T) and on the transference information (R) of the duplicated cards. Even

though R could be forged using previous cards from the user, T is computed using the player's private key (and is bound to R, so that a forged R will relate to no transference). hence, for adversaries to create two coins that will blame some player, they have to either corrupt the player (but, then, all the cards from this player already become the adversary's) or to be able to create a forged card (i.e., forge the underlying proof of knowledge, which should be unfeasible if the underlying protocol is secure).

5.2 Treating an illegal duplication

While all secure electronic cash schemes provide a method for identifying a double-spending, they seldomly discuss what to do when it is found. Commonly, the actions to be taken when the fraud is found is considered an administrative measure for the implementation, and depends on legislative sanction. For example, the bank might decide to charge the culprits into paying what they have illegally created (and take upon itself the costs when charging the adversaries is not possible) or refuse to pay for duplicated coins. However, it is important to bear in mind that, unless the bank has some mechanism to detect that the user presenting a duplicated coin is in collusion with the duplicator, refusing a coin may be illegal, as the bank is appropriating the user's assets.

Even though P2P TCGs do not deal with cash directly, this problem is likely even more complex in this scenario. In some TCGs, some cards are produced in a limited amount, and for this reason are called *rare* cards. These cards usually have powerful effects in the game or have some artwork that reflects the limited production, which ends up increasing their value for collectors. If the market decides to keep the cards, accepting duplicated ones, an attacker may multiply a rare card with the sole objective of flooding the system with such a card. The game server must, thus, have an alternative to avoid this undertaking. Obliging the transferences to occur in the presence of a trusted party is an option, but it would go against the requirement of having TTP-free

transferability. In paid games, it is also possible to charge the duplicator accordingly, making the duplication attempts to expensive for attackers. Another way to provide this balance is to compel the owners rare cards to refresh them before using or trading them to somebody else. This can be easily done if the owner of the card, $\mathfrak{P}$, proves that all ownership tags of the cards refers to him/herself, i.e., for a card with h tags in Π_T, $\bigwedge_{j=0}^{h} \mathcal{V}_r(pk_P, r_j, \phi_{r_j}) \stackrel{?}{=} 1$.

For *common* cards, the game server may follow the loose restriction of keeping both cards, since increasing the number of these cards may not disrupt the fluctuation of card values. Nonetheless, the registration center may need a way to unregister the duplicators from the system as a punishment for their misbehavior. For example, it can set a short *expiration date* to the user certificate, forcing the players to renovate their key certificate, and denying further subscription of duplicators, or simply have a revocation list that should be checked by the players periodically, as usually occurs with certificates on the Internet.

5.3 Performance analysis

Aiming to evaluate the performance of the proposed scheme, we benchmarked it using an efficient library for elliptic curve cryptography: RELiC (ARANHA; GOUVÊA, —). This library provides efficient methods for the operations necessary in the proposed protocol: operations in finite field and in elliptic curve points, hash functions, pairings computation, and efficient storage for points. The implementation supports presets with a high performance in 64-bit architectures (ARANHA et al., 2011) (for the game server), and for mobile compatibility (e.g., ARM architectures (ARAUJO, 2013), for the players' devices) when working with a 128-bit security level pairing-friendly curve. We summarize the compiling parameters employed in our implementation in Appendix B, where we also present the curve parameters and additional information related to their execution (e.g., hash function and pseudorandom generator).

Table 6: Size (in bytes) to represent group elements using RELiC toolkit with 128 security bits

Group element	Expanded	Compressed
$\mathbb{Z}_q$	32	
$\mathbb{G}_1$	65	33
$\mathbb{G}_2$	129	65
$\mathbb{G}_T$	384	256

Source: Author

To store the elements that compose the cards, we need to represent elements in $\mathbb{Z}_q$, $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$. RELiC stores them as bit arrays, whose lengths depend on the group order. Elliptic curve points can be serialized using one out of two possible representations: compressed and expanded. Compressed points are usually used for transmitting them to other devices, or for storing values that are more rarely used, such as each individual card. Expanded points are used for values frequently used, such as the curve generator or the market public key. The lengths of the representations are presented in Table 6.

The processing time is computed in the pairing setting mode. We execute operations in the finite field (scalar addition – FFA – and scalar multiplication – FFM), in the pairing source groups (elliptic curve addition – ECA – and elliptic curve multiplication – ECM, in $\mathbb{G}_1$ or $\mathbb{G}_2$), in the pairing target group (extended field multiplication – FXM – and extended field exponentiation – FXE), and the pairing computation (PC). Table 7 presents these values for an Intel Core i7 4790, with 3.6 GHz without Hyper Threading (HT) and 4 GHz with HT.

The timing for each operation and the following timing tests were executed 100 times. The sample size was devised so that the standard deviation did not surpass 10% of the mean value. The largest standard deviation was 9.57% of the mean value, to prove knowledge of a signature (ϕ_σ).

Table 7: Processing time of group operations, on a 4 GHz with Hyper Threading (HT) or 3.6 GHz without it

Group	Operation	Operation (without HT)		Operation (with HT)	
		Cycles	Time	Cycles	Time
$\mathbb{Z}_q$	FFA	111	0.031 μs	114	0.029 μs
	FFM	1399	0.39 μs	1408	0.35 μs
$\mathbb{G}_1$	ECA	4267	1.19 μs	4270	1.10 μs
	ECM	76559	0.021 ms	776251	0.19 ms
$\mathbb{G}_2$	ECA	11850	0.003 ms	11742	0.003 ms
	ECM	1811813	0.50 ms	1811097	0.45 ms
$\mathbb{G}_T$	FXM	15637	0.004 ms	15643	0.004 ms
	FXE	3455933	0.96 ms	3483206	0.87 ms
	PC	4679812	1.30 ms	4713669	1.19 ms

Source: Author

Table 8: Size to represent proofs of knowledge, with 128 security bits

Proof	Elements to prove		Equations	Proof Size		
	in $\mathbb{G}_1$	in $\mathbb{G}_2$		$\mathbb{G}_1$	$\mathbb{G}_2$	Bytes
ϕ_{UID}	2	5	6	28	34	3134
ϕ_σ *	24	4	18	120	80	9160
ϕ_{T_j}	4	7	8	40	46	4310
ϕ_{r_j}	5	8	10	50	56	5290

* Proof for signing 4 messages: the first owner's private key sk_{P_0}, the class identifier CID, the serial seed s and the transference seed t.

Source: Author

5.3.1 Storage

To store a card $\mathfrak{card} = (UID, CID, \phi_{UID}, \phi_\sigma, \Pi_T = \{T_j, \phi_{T_j}, r_j, \mathfrak{info}_j\}_{j=0...h})$ that was used or traded h times, we have to store all elements that are represented. The unique identifier (UID) and each transference and ownership tags (respectively, $T_j, r_j \in \Pi_T$) are elements in $\mathbb{G}_1$; they are output from VRFs, and, hence, $1 + 2h$ elements are necessary. The class identifier (CID) is a numeric representation of the class and can be represented by one element in $\mathbb{Z}_q$, whereas the public information $\mathfrak{info}_j$ can be any string of bytes (for simplicity, we consider it here as another element in $\mathbb{Z}_q$). Each proof has different sizes, because they depend on the proved values and the PPE defined for them. Table 8 summarizes the costs for the proof of construction of UID (ϕ_{UID}), T_j (ϕ_{T_j}), and r_j (ϕ_{r_j}), and for the proof of signature from the market (ϕ_σ).

Table 9: Assemble of costs to store a card, with 128 security bits

$\mathfrak{card}$	UID	ϕ_{UID}	CID	ϕ_σ	T	ϕ_T	r	$\mathfrak{info}$
$\mathbb{Z}_q$			1					1
	1				1			
$\mathbb{G}_1$	1	28		120	1	40	1	
	149				42			
$\mathbb{G}_2$		34		80		46		
	114				46			
bytes	33	3134	32	9160	33	4310	33	32
	12359				4408			

Combining all costs, as illustrated in Table 9, we deduce the expression:

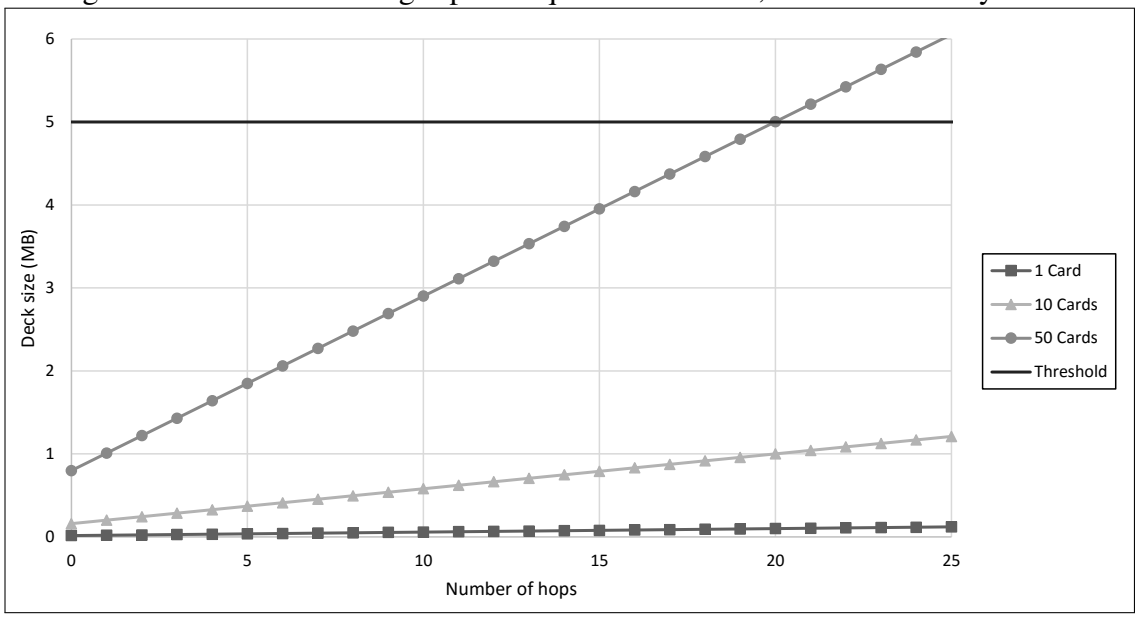
$$c(h) = \begin{pmatrix} 1(\mathbb{Z}_q) \\ 149(\mathbb{G}_1) \\ 114(\mathbb{G}_2) \end{pmatrix} + \begin{pmatrix} 1(\mathbb{Z}_q) \\ 42(\mathbb{G}_1) \\ 46(\mathbb{G}_2) \end{pmatrix} h = 12359 + 4408h \; bytes \tag{5.1}$$

Since RELiC stores all elements in constant-size variables, this cost was verified using the library. This expression is also used for cards being transmitted.

As aforementioned, the card grows for each usage or trade. The growth is linear with each hop, as new T_j, ϕ_{T_j}, r_j and $\mathfrak{info}_j$ are created. For a single card the growth is not so noticeable. However, if we consider that the players' decks have around 50 cards, as it is common in commercial TCGs, the storage costs become more relevant, as illustrated in Figure 45.

In the chart from Figure 45, we set a threshold limit of 5 MB to store security information related to a deck, which is equivalent to approximately 19 hops. This is an admissible value if we consider that a game may need more than 100 times more storage space for data related to the game (functionality and graphical designs): in Hearthstone, for example, the minimum required storage space is 2 GB for a mobile device with Android OS (BLIZZARD ENTERTAINMENT, —). In this case, 5 MB for security measures is equivalent to only 0.25% of the game data.

Figure 45: Growth of storage space required for a deck, with 128 security bits



Source: Author

5.3.2 Communication

Almost all protocol methods are interactive, and require communication between the involved parties. The underlying e-cash protocol was constructed for minimizing the number of messages exchanged for each method, so the only method that is not executed with one request and one response is the *Refresh* protocol. Also, without loss of generality, the verification step and the stamping step can happen simultaneously, reducing the number of interactions: after all, the fresh card ought to be returned to the player if and only if the verification process is successful. In the following, we present the number of group elements for each message and summarize these values in Table 10, with the corresponding size in bytes using RELiC.

For the *Register* protocol, a player $\mathfrak{P}$ sends a message with three parts: his/her identity id_P, his/her public key pk_P, and the proof that he/she knows the associated private key ϕ_P. The registration center responds with the certificate σ_P if the subscription is accepted. We can represent id_P as a numeric value in $\mathbb{Z}_q$ and the public key as an element in $\mathbb{G}_T$. The proof of knowledge is a proof of 2 PPEs, with 1 variable in $\mathbb{G}_1$ and

2 in $\mathbb{G}_2$ (which results in 10 elements in $\mathbb{G}_1$ and 12 in $\mathbb{G}_2$). This adds up to 1 element in $\mathbb{Z}_q$, 10 in $\mathbb{G}_1$, 12 $\mathbb{G}_2$ and 1 in $\mathbb{G}_T$ for the complete request. From Table 5, independently from the number of messages, we need 5 elements in $\mathbb{G}_1$, 1 in $\mathbb{G}_2$ and 1 element in $\mathbb{Z}_q$ to represent a signature with the P-signature scheme presented in Section 3.8, which represents the entire response message.

For the *Stamp* protocol, the player sends the commitment K_s for the blind signature, the proof of construction of the commitment ϕ_{K_s}, and the class identifier CID of the card to buy. The market responds with the partial signature σ' and the partial serial seed component s''. The commitment is an element in $\mathbb{G}_1$ and the CID is a value in $\mathbb{Z}_q$. The proof of the commitment for 4 messages (player's private key sk_P, serial seed s', transference seed t and an empty space for the class identifier CID) is composed by 12 PPEs with 16 variables in $\mathbb{G}_1$, resulting in 80 elements in $\mathbb{G}_1$ and 48 in $\mathbb{G}_2$. The signature, independently of the number of messages, contains 5 elements in $\mathbb{G}_1$, 1 in $\mathbb{G}_2$ and 1 in $\mathbb{Z}_q$. Thus, the request message is formed by 1 element in $\mathbb{Z}_q$, 81 elements in $\mathbb{G}_1$ and 48 elements in $\mathbb{G}_2$, while the response comprises 2 elements in $\mathbb{Z}_q$, 5 in $\mathbb{G}_1$ and 1 in $\mathbb{G}_2$.

For the *Send* protocol, the receiver sends the public and private information of the trade ($\mathfrak{info}$ and r), and the proof ϕ_r that he/she knows the private key of r. After computing the new hop, the sender responds with the proof of ownership ϕ_{r_h} that he/she is the current owner, before passing the card $\mathfrak{card}'$ to the new owner. The public information $\mathfrak{info}$ is represented as a numeric value in $\mathbb{Z}_q$, and the private information r, as a value in $\mathbb{G}_1$. Both proofs, as presented in Table 8, are represented by 50 elements in $\mathbb{G}_1$ and 56 elements in $\mathbb{G}_2$. Finally, the card has the cost associated with Equation 5.1. Thereby, the receiver sends to the sender 1 elements in $\mathbb{Z}_q$, 51 in $\mathbb{G}_1$ and 56 in $\mathbb{G}_2$, that responds with 50 elements in $\mathbb{G}_1$, 56 in $\mathbb{G}_2$, and a card ($(1 + h)$ elements in $\mathbb{Z}_q$, $(149 + 42h)$ in $\mathbb{G}_1$ and $(114 + 46h)$ in $\mathbb{G}_2$).

The *Play* protocol involves only one message from the sender. He/She sends the

card $\mathfrak{card}'$ to play, and two proofs of ownership (r_h and r_{h+1}). The proofs are each represented by 50 elements in $\mathbb{G}_1$ and 56 elements in $\mathbb{G}_2$ (see Table 8), and the card has the cost associated with Equation 5.1. For a deck of 50 cards (and 2 proofs of ownership for each card), the player sends $(50 + 50h)$ elements in $\mathbb{Z}_q$, $(7550 + 2100h)$ in $\mathbb{G}_1$ and $(5812 + 2300h)$ in $\mathbb{G}_2$.

In the *Report* protocol, the player sends an entire deck of cards to the game auditor, so the costs are a multiple of the costs of a card. For example, for a deck with 50 cards, the player sends $(50 + 50h)$ elements in $\mathbb{Z}_q$, $(7450 + 2100h)$ in $\mathbb{G}_1$ and $(5700 + 2300h)$ in $\mathbb{G}_2$.

Finally, the *Refresh* protocol is a combination of the *Send* and *Stamp* protocols, and have their costs combined. This results in $(1 + h)$ elements in $\mathbb{Z}_q$, $(249 + 42h)$ in $\mathbb{G}_1$ and $(226 + 46h)$ in $\mathbb{G}_2$ sent to the server, which responds with 2 elements in $\mathbb{Z}_q$, 5 in $\mathbb{G}_1$ and 1 in $\mathbb{G}_2$.

Table 10 presents the costs in numbers (considering a deck with 50 cards, whenever applicable). These numbers are also illustrated in Figure 46, in which the messages for each protocol is presented when the cards are not sent. For the methods that depend on how many times a card was used or traded, Figure 47 also illustrates the growth of the message sent if the entire deck is at the same hop, in which each color degree represents how much the message grows when the card has been used or sent.

Since the presented protocol is intended for P2P architecture, it is expected that the players will connect via an ad-hoc wireless network, using, for example, Bluetooth®. Bluetooth® connection is efficient for this kind of operation, since the protocol is capable of transmitting data from 720 kbps up to 54 Mbps if the devices are in line of sight (BLUETOOTH SIG, 2014). For example, for cards with 10 hops, the slower connection will take less than 200 ms for sending a card, and around 8 seconds for playing with the entire deck. This transmission might be performed before the match begins, when both players must send their decks to one another, in which case the setup time

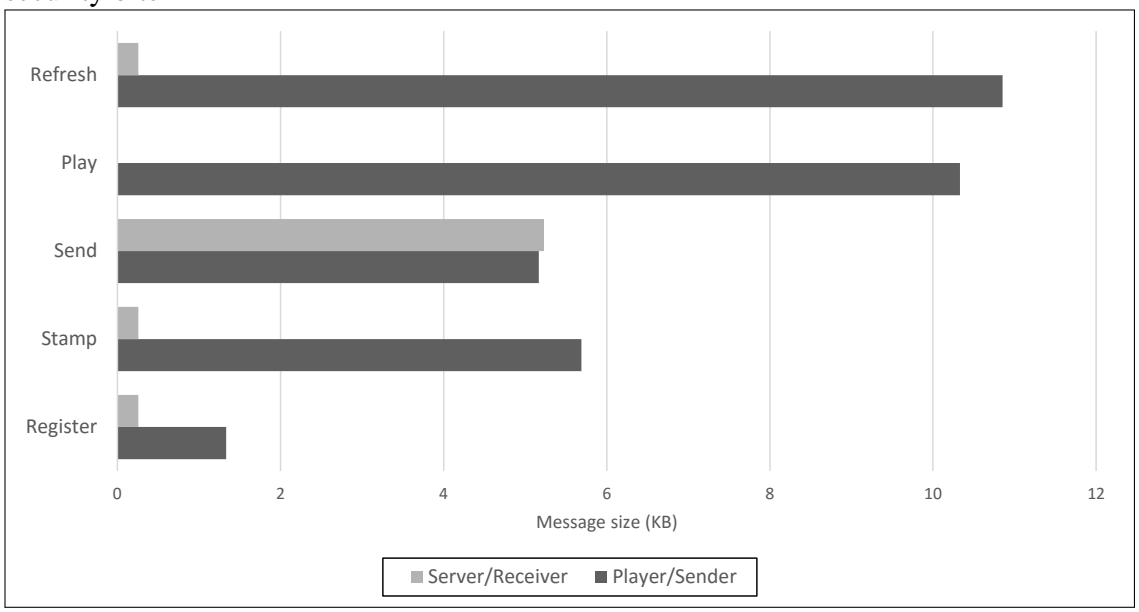
Table 10: Communication cost per protocol, considering decks with 50 cards and 128 security bits

Protocol	Player			
	$\mathbb{Z}_q$	$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{G}_T$
Register	1	10	12	1
Stamp	1	81	48	0
Send *	1 $1 + h$	51 $199 + 42h$	56 $170 + 46h$	0 0
Play	$50 + 50h$	$7550 + 2100h$	$5812 + 2300h$	0
Report	$50 + 50h$	$2100h$	$2300h$	0
Refresh	$1 + h$	$249 + 42h$	$226 + 46h$	0
Protocol	Server			
	$\mathbb{Z}_q$	$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{G}_T$
Register	1	5	1	0
Stamp	2	5	1	0
Refresh	2	5	1	0

* Values for Receiver/Sender

Source: Author

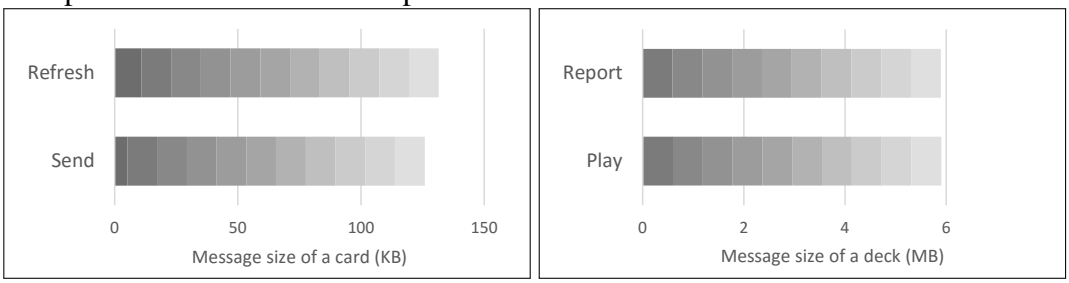
Figure 46: Communication cost per protocol, considering decks with 50 cards and 128 security bits



Source: Author

is approximately 20 s; alternatively, the players can simply send the cards' information that are strictly required by the application before the match begins, and only later send the security information related to them for allowing the verification to occur. Even in the worst scenario, 20 seconds it likely to be considered an acceptable time in a game

Figure 47: Growth of message size per hop of card or 50-dard deck, each color degree corresponds to one additional hop



Source: Author

Figure 48: Processing time for constant methods, with 128 security bits

Source: Author

in which the players usually prepare tables, markers and even dice before a match, depending on the players' cards.

5.3.3 Processing time

The execution time for the protocol is likely dominated by the pairing computations, even though the number of operations in the pairing setting are also cumbersome. We have analyzed using benchmarks for the RELiC library in a desktop computer, where the processing power is usually higher than in a mobile device.

Some methods have small variations in the processing time to compute them. They depend only on their operations and not on the operands passed, and we call them *constant* methods. On the server side, the constant methods are: Setup, Register, Stamp and Identify. On the players' side, they are: Register, Stamp, Send (sender-side), Play (prover-side) and Refresh. The mean execution time is presented in Figure 48.

The methods Setup, Identify and Register (player-side) are the fastest methods, for

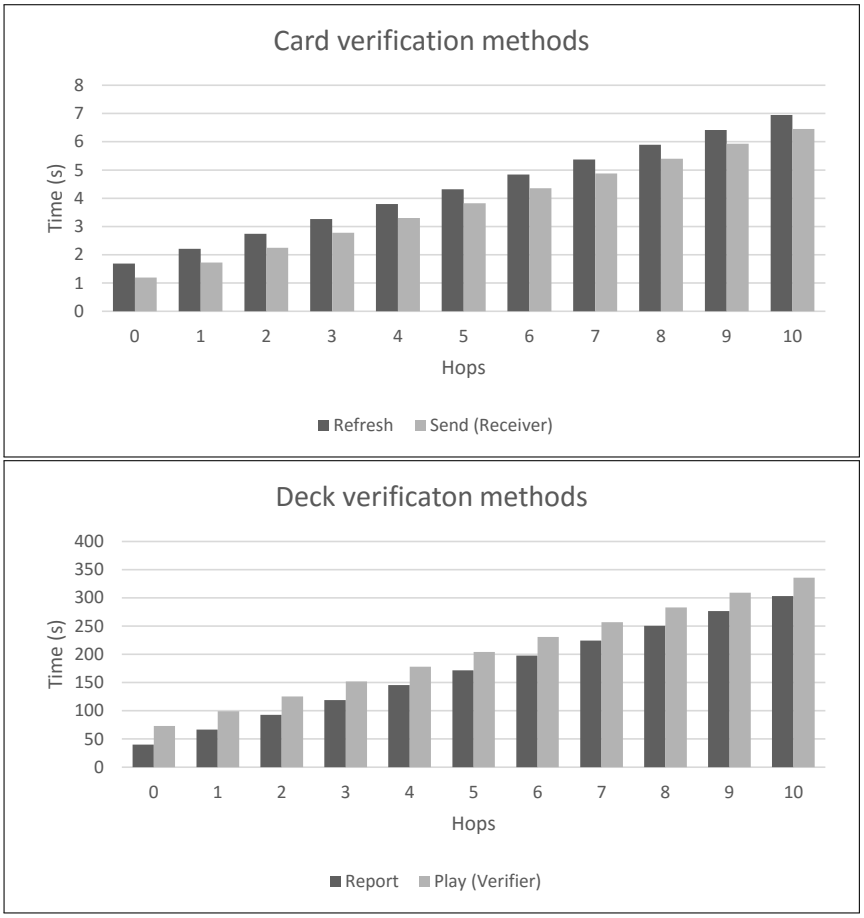
they involved a few and fast operations to compute, so they are computed in less than 20 ms. The methods that include proofs of knowledge to the card (Stamp, Send, Play and Refresh) are the most expensive ones. The verifier of these proofs is the entity that ends up with a higher load, because they have to compute bilinear pairings and operations in $\mathbb{G}_T$, while the prover computes operations in the source groups.

The remaining methods also depend on the size of the card, because they have to verify a large number of proofs of knowledge. These *variable* methods also limit the size of the card, due to the fact that bigger cards will have longer verification time. On the server side, the variable methods are Report and Refresh, while on the players' side they are Send (receiver-side) and Play (verifier-side). Figure 49 illustrates the mean execution time of such methods as a card is passed along. For better readability, the charts are divided in methods that verify one card or the entire deck.

The most expensive methods are those that verify an entire deck: the server-side of Report and the verifier-side of Play. For a complete deck of 50 cards that has been used 5 times, the verification is complete in less than 3 minutes. If a mobile device is used instead of a desktop, the times are likely to be higher. For example, the comparison between the works of Aranha et al. (2011) and of Araujo (2013) shows that a legacy mobile device (Motorola Milestone 1, ARMv6 600 MHz) is around 10 times slower to compute a pairing than a desktop with a Intel Core i5 1,6 GHz. Hence, the protocol would take around 30 min to complete the execution in this device.

All in all, we can argue that these timings are quite reasonable for common trading card games, even if played with legacy devices. The reason is that the preparation of a deck can be done beforehand, much before the match starts; in addition, the verification of the corresponding proofs of knowledge can happen in background during matches, which usually take several minutes. Therefore, in practice those costs can be made transparent to players, in especial if the game is played using modern mobile devices, whose processing capabilities are becoming more and more comparable with those of

Figure 49: Processing time for variable methods, considering a 50-card deck and 128 security bits



Source: Author

desktop computers.

5.4 Offload methods

In Section 5.3 we discussed the costs for storage, communication and execution of our card trading protocol. Even though they are compliant for usability, some computations are still expensive for devices with low computational power, for example, for a mobile device to verify an entire deck. Some policy methods can reduce these costs, at some cost for an attacker to take advantages of the system. We now describe some methods to provide more usability, and present how much peril they offer to security.

5.4.1 Delegate deck verification

Before two players will play in a match, they execute the *Play* protocol for every card in their play deck; during this process, one proves the correctness of each card in the deck and the other verifies the proofs. Proving is faster than verifying, mainly because verification is executed for every hop in the cards while proof depends on the last hop (see Section 5.3 for details).

In the discussion of the previous sections, we proposed that all cards are verified completely. Nevertheless, an alternative is to have the player delegate this process to the game auditor. More precisely, if a player verifies only the ownership of each card, he/she reduces the costs in more than 500 ms per hop. Then, that player can execute the *Report* protocol to send the entire deck to the game auditor, which verifies its correctness completely. If the verification holds, there was no problem in the system. However, if the prover had forged the cards with which it is playing, the game auditor can inform the verifier that the cards were malformed. The verifier can then flag the prover so, the next time they play together, the entire deck will be verified offline to guarantee integrity.

It is important to notice that such process is not totally foul proof: unfortunately, the game server cannot blame the said dishonest player (e.g., downgrading his/her reputation), because it cannot provide any proof to guarantee exculpability. After all, the verifier may have modified the report to harm an honest player. It can, nevertheless, consider the match invalid and wait for the corresponding report from the prover to check for errors.

5.4.2 Reuse match information

To play a match, a player executes the *Play* protocol to create an auto-transference of each card in the play deck. If one desires to play several consecutive matches with

the same cards, in principle it would be necessary to create new proofs for each match. However, if the opponent consents, that player may actually reuse an existing proof for several consecutive matches, saving processing time and storage.

Usually, the public information $\mathfrak{info}_j$ used to compute R_j may present the timestamp to avoid replay attacks and guarantee that the proof was constructed for that transference. The usage/trade is to be refused if the timestamp is out of a predefined time window. However, if the adversary does not change from a match to another, there is no problem in replaying the last card. If the game server finds several reports of the same card, it can discard the recurrences and maintain the report of only one usage. If the opponents vary from match to match, the new one should refuse a card with a timestamp from outside the acceptable time window. Otherwise the player may try to replay a card that does not belong to him/her anymore with the preparation for a previous match he/she had played. Since the card was prepared before the trade, the game server would not be able to identify the cheat of the type *trade-then-play*.

5.5 Summary

In this Chapter we have analyzed the protocol, both qualitatively and quantitatively. We have discussed how the protocol fulfills the security properties defined in Section 2.3, and how one implementing a real application using this protocol could handle the illegal duplication of cards. We have also analyzed the computational costs of the protocol, in communication, storage and processing time. We have concluded the analysis discussing methods to lower costs to more constrained devices with minor security concerns that do not break the security properties required.

6 CONCLUSIONS

Given the success of TCGs and the growth of casual and social digital games, it is important to consider the usage of protocols that allows users to securely trade cards. In this work, we have expanded the works of SecureTCG (SIMPLICIO JR. et al., 2014) and Match+Guardian (PITTMAN; GAUTHIERDICKEY, 2013), that provide in-game cheating detection, but had left as future work the construction of a method to trade cards in P2P architecture. The scenario described in Chapter 2 introduces that a central server is not necessary during trading or playing. It works as a central authority to stamp new cards and register players, aiming to allow the detection of irregularities when auditing is necessary. Playing and trading can happen without the TTP, whilst players are still able to verify the validity of cards they are receiving and of cards their opponents use in a match. The requirements we have proposed address the transferability required for P2P trading, while providing balance to the system. If the balance is broken, the game server is capable of identifying the transgressor and takes the necessary measures to ensure security.

As the requirements for securely trading cards in TCGs are similar to the ones of transferable e-cash, the proposed protocol was based on an efficient and modular compact e-cash scheme (BELENKIY et al., 2009). The basis to this protocol, presented in Chapter 3, focus on zero-knowledge proofs of knowledge to provide transferability to the system. Even though using Groth-Sahai proofs (GROTH; SAHAI, 2008) are quite efficient, they are still the bottleneck to the system because they rely on computationally expensive operations: bilinear pairings. To comply with this method, we

have used and adapted the P-signature scheme from Izabachène, Libert and Vergnaud (2011). This protocol had suffer a great loss of security level due to attacks on the building block of symmetric pairings. However, we were able to circumvent this issue by converting this signature to an asymmetric pairing setting, together with the security reductions to hard computational problems. The resulting protocol was derived selecting the largest set of elements in $\mathbb{G}_1$, whose operations are faster and the memory representation is more efficient than that of elements in $\mathbb{G}_2$.

The requirements are fulfilled by the protocol proposed, which is described in detail in Chapter 4. After the game server has the game set up, the players can register, so they give enough information to be identified if they cheat. The game market is responsible for providing new cards to the game, in which the stamping is anonymous and can be verified offline. Without a TTP, the players can play with their cards or trade them, relinquishing the ownership to a new player. To avoid the problem of indefinite growth of cards (inherent to e-cash), players can refresh their cards for computationally cheaper ones (in storage size and computational power to verify). The refreshed cards, together with reported information provided by the players, are sufficient to the game auditor to identify cheaters (illegal duplicators).

The proposed protocol is quite efficient and can be used in practice. A complete deck of 50 cards require a few MB depending on how many times the cards had been used or passed along. For example, a deck with 10 hops requires (before the next refresh) around 3 MB of storage space (equivalent to 0.15% of the game data of Hearthstone, an example of commercial TCG). Using the same deck in a match, where the devices are connected via Bluetooth®, transferring the information will take less than 10 seconds to send the deck to an opponent. In processing time, the same methods are also the most expensive ones. Verifying an entire deck takes around 5 minutes in a desktop and may reach 30 minutes in a legacy mobile device. Albeit this time would usually be impracticable, the verification can be executed in background during the

match, that usually takes several minutes, and the costs are made transparent to the players. Also, it is possible to delegate the verification the the game server, that can identify illegal duplications on behalf of players with less powerful devices.

6.1 Future work

We leave as future work the analysis in mobile devices to further validate the efficiency of e-cash protocols, and their application in the TCG scenario. This analysis can provide evidence of the evolution of pairing computation on restricted devices, comparing with the evolution of mobile devices, and possibilities to develop applications in real TCGs and in e-cash.

Further research can also focus on improvements in efficiency and privacy. For example, the work of Baldimtsi et al. (2015), that uses malleable signatures based on Groth-Sahai proofs (CHASE et al., 2014), has recently opened an area for complex protocols with higher privacy. Even though full anonymity is not an essential property for trading cards, applying malleable signatures to homomorphic encryption with little loss of efficiency could benefit both TCGs and e-cash protocols.

6.2 Publications

As direct or indirect result of the research carried out during this thesis, we produced the following publications:

- Conference Paper: in (SILVA; SIMPLICIO JR., 2015), we describe the construction of the proposed construction in this work. It has received the award of "Best Paper" (among 77 submissions).
- Journal Article: in (SIMPLICIO JR. et al., 2016), we propose a authenticated key-agreement for Internet of Things. We provide an escrow-free protocol with

implicit certificates to obtain a lightweight protocol and evaluates its performance and security, comparing our results with existing solutions, and showing that some very efficient protocols are actually flawed.

REFERENCES

ABE, M. et al. Constant-size structure-preserving signatures: Generic constructions and simple assumptions. In SPRINGER. *Advances in Cryptology (ASIACRYPT'12)*. Beijing, China, 2012. p. 4–24.

____. Structure-preserving signatures and commitments to group elements. In SPRINGER. *Advances in Cryptology (CRYPTO'10)*. Santa Barbara, USA, 2010. p. 209–236.

____. Converting cryptographic schemes from symmetric to asymmetric bilinear groups. In SPRINGER. *Advances in Cryptology (CRYPTO'14)*. Santa Barbara, USA, 2014. p. 241–260.

ARANHA, D. F.; GOUVÊA, C. P. L. *RELIC is an Efficient LIbrary for Cryptography*. —. Available from Internet: <https://github.com/relic-toolkit/relic>. Cited 11/08/2016.

ARANHA, D. F. et al. Faster explicit formulas for computing pairings over ordinary curves. In SPRINGER. *Advances in Cryptology (EUROCRYPT'11)*. Tallinn, Estonia, 2011. p. 48–68.

ARAUJO, R. W. M. *Autenticação e comunicação segura em dispositivos móveis de poder computacional restrito*. MSc Thesis (MSc) — Universidade de São Paulo, Instituto de Matemática e Estatística, São Paulo, Brazil, 2013.

BALDIMTSI, F. et al. Anonymous transferable e-cash. In SPRINGER. *Practice and Theory in Public-Key Cryptography (PKC 2015)*. Gaithersburg, USA, 2015. p. 101–124.

BALLARD, L. et al. *Correlation-resistant storage*. Baltimore, USA, 2005. (TR-SP-BGMM-050507).

BARBULESCU, R. et al. A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In SPRINGER. *Advances in Cryptology (EUROCRYPT'14)*. Copenhagen, Denmark, 2014. p. 1–16.

BELENKIY, M. et al. P-signatures and noninteractive anonymous credentials. In SPRINGER. *5th Theory of Cryptography Conference*. New York, USA, 2008. p. 356–374.

____. Compact e-cash and simulatable vrfs revisited. In SHACHAM, H.; WATERS, B. (Ed.). *Pairing-Based Cryptography (Pairing'09)*. [S.l.]: Springer, 2009. p. 114–131. ISBN 978-3-642-03297-4.

BLIZZARD ENTERTAINMENT. *Hearthstone system requirements*. —. Available from Internet: <https://us.battle.net/support/en/article/hearthstone-system-requirements>. Cited 11/08/2016.

BLUETOOTH SPECIAL INTEREST GROUP. *Specification of the Bluetooth® System*. [S.l.], 2014. Available from Internet: <https://www.bluetooth.org/en-us/specification/adopted-specifications>. Cited 11/08/2016.

BLUM, M.; FELDMAN, P.; MICALI, S. Non-interactive zero-knowledge and its applications. In ACM. *20th annual ACM symposium on theory of computing*. Chicago, USA, 1988. p. 103–112.

BONEH, D. The decision diffie-hellman problem. In SPRINGER. *3th international symposiun on algorithmic number theory*. Portland, Oregon, 1998. p. 48–63.

BONEH, D.; GENTRY, C.; WATERS, B. Collusion resistant broadcast encryption with short ciphertexts and private keys. In SPRINGER. *Advances in Cryptology (CRYPTO'05)*. Santa Barbara, USA, 2005. p. 258–275.

BOYEN, X.; WATERS, B. Full-domain subgroup hiding and constant-size group signatures. In SPRINGER. *Practice and Theory in Public-Key Cryptography (PKC 2007)*. Beijing, China, 2007. p. 1–15.

CAMENISCH, J.; HOHENBERGER, S.; LYSYANSKAYA, A. Compact e-cash. In SPRINGER. *Advances in Cryptology (Eurocrypt'05)*. Aarhus, Denmark, 2005. p. 302–321.

CAMENISCH, J.; LYSYANSKAYA, A. A signature scheme with efficient protocols. In SPRINGER. *3rd international conference on security in communication networks*. Amalfi, Italy, 2003. p. 268–289.

____. Signature schemes and anonymous credentials from bilinear maps. In SPRINGER. *Advances in Cryptology (CRYPTO'04)*. Santa Barbara, USA, 2004. p. 56–72.

CANARD, S.; GOUGET, A. Anonymity in transferable e-cash. In SPRINGER. *Applied Cryptography and Network Security (ACNS)*. New York, USA, 2008. p. 207–223.

CANARD, S.; GOUGET, A.; TRAORÉ, J. Improvement of efficiency in (unconditional) anonymous transferable e-cash. In TSUDIK, G. (Ed.). *Financial Cryptography and Data Security*. [S.l.]: Springer, 2008. p. 202–214. ISBN 978-3-540-85229-2.

CANETTI, R.; GOLDREICH, O.; HALEVI, S. The random oracle methodology, revisited. *Journal of the ACM (JACM)*, ACM, New York, USA, vol. 51, no. 4, p. 557–594, 2004.

CHASE, M. et al. Malleable proof systems and applications. In SPRINGER. *Advances in Cryptology (Eurocrypt'12)*. Cambridge, UK, 2012. p. 281–300.

____. Malleable signatures: Complex unary transformations and delegatable anonymous credentials. IACR Cryptology ePrint Archive. 2013. Available from Internet: <http://eprint.iacr.org/2013/179.pdf>. Cited 11/08/2016.

____. Malleable signatures: New definitions and delegatable anonymous credentials. In IEEE. *2014 IEEE 27th Computer Security Foundations Symposium (CSF)*. Vienna, Austria, 2014. p. 199–213.

CHATTERJEE, S.; MENEZES, A. On cryptographic protocols employing asymmetric pairings – the role of ψ revisited. *Discrete Applied Mathematics*, Elsevier, vol. 159, no. 13, p. 1311–1322, 2011.

CHAUM, D. Blind signatures for untraceable payments. In SPRINGER. *Advances in cryptology (CRYPTO'82)*. Santa Barbara, USA, 1983. p. 199–203.

CHAUM, D.; EVERTSE, J.-H.; GRAAF, J. V. D. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In SPRINGER. *Advances in Cryptology (EUROCRYPT'87)*. Amsterdam, The Netherlands, 1988. p. 127–141.

CHAUM, D.; PEDERSEN, T. P. Transferred cash grows in size. In SPRINGER. *Advances in Cryptology (Eurocrypt'92)*. Balatonfüred, Hungary, 1993. p. 390–407.

____. Wallet databases with observers. In SPRINGER. *Advances in Cryptolog (CRYPTO'92)*. Santa Barbara, USA, 1993. p. 89–105.

DIFFIE, W.; HELLMAN, M. E. New directions in cryptography. *Information Theory, IEEE Transactions on*, IEEE, vol. 22, no. 6, p. 644–654, 1976.

DODIS, Y.; YAMPOLSKIY, A. A verifiable random function with short proofs and keys. In SPRINGER. *Practice and Theory in Public-Key Cryptography (PKC 2005)*. Les Diablerets, Switzerlands, 2005. p. 416–431.

ELGAMAL, T. A public key cryptosystem and a signature scheme based on discrete logarithms. In SPRINGER. *Advances in Cryptology (CRYPTO'84)*. Santa Barbara, USA, 1984. p. 10–18.

ENTERTAINMENT SOFTWARE ASSOCIATION. *Essential facts about the computer and video game industry*. 2015. Available from Internet: <http://www.theesa.com/wp-content/uploads/2015/04/ESA-Essential-Facts-2015.pdf>. Cited 11/08/2016.

FEIGE, U.; SHAMIR, A. Witness indistinguishable and witness hiding protocols. In ACM. *22nd annual ACM symposium on theory of computing*. Baltimore, USA, 1990. p. 416–426.

FIAT, A.; SHAMIR, A. How to prove yourself: Practical solutions to identification and signature problems. In SPRINGER. *Advances in Cryptology (CRYPTO'86)*. Santa Barbara, USA, 1987. p. 186–194.

FUCHSBAUER, G.; POINTCHEVAL, D.; VERGNAUD, D. Transferable constant-size fair e-cash. In SPRINGER. *8th International Conference on Cryptology and Network Security*. Kanazawa, Japan, 2009. p. 226–247.

GALBRAITH, S. D.; PATERSON, K. G.; SMART, N. P. Pairings for cryptographers. *Discrete Applied Mathematics*, Elsevier, vol. 156, no. 16, p. 3113–3121, 2008.

GAUTHIERDICKEY, C.; RITZDORF, C. Secure peer-to-peer trading for multiplayer games. In IEEE. *2012 11th Annual Workshop on Network and Systems Support for Games (NetGames)*. Venice, Italy, 2012. p. 1–6.

____. Secure peer-to-peer trading in small-and large-scale multiplayer games. *Multimedia Systems*, Springer Berlin Heidelberg, vol. 20, no. 5, p. 595–607, 2014.

GOLDWASSER, S.; KALAI, Y. T. On the (in) security of the fiat-shamir paradigm. In IEEE. *44th Annual IEEE Symposium on Foundations of Computer Science, 2003*. Crambridge, UK, 2003. p. 102–113.

GROTH, J.; OSTROVSKY, R.; SAHAI, A. Non-interactive zaps and new techniques for nizk. In SPRINGER. *Advances in Cryptology (CRYPTO'06)*. Santa Barbara, USA, 2006. p. 97–111.

____. Perfect non-interactive zero knowledge for np. In SPRINGER. *Advances in Cryptology (EUROCRYPT'06)*. Saint Petersburg, Russia, 2006. p. 339–358.

GROTH, J.; SAHAI, A. Efficient non-interactive proof systems for bilinear groups. In SPRINGER. *Advances in Cryptology (Eurocrypt'08)*. Istanbul, Turkey, 2008. p. 415–432.

HANKERSON, D.; MENEZES, A. J.; VANSTONE, S. *Guide to Elliptic Curve Cryptography*. New York, USA: Springer, 2004. ISBN 0-387-95273-X.

HUSEMÖLLER, D. *Elliptic curves*. 2. ed. New York, USA: Springer, 2004. ISBN 978-0-387-21577-8.

IZABACHÈNE, M.; LIBERT, B.; VERGNAUD, D. Block-wise P-signatures and non-interactive anonymous credentials with efficient attributes. In SPRINGER. *13th IMA International Conference on Cryptography and Coding*. Oxford, UK, 2011. p. 431–450.

KOBLITZ, N. Elliptic curve cryptosystems. *Mathematics of computation*, American Mathematical Society, vol. 48, no. 177, p. 203–209, 1987.

LENSTRA JR, H. W. Factoring integers with elliptic curves. *Annals of mathematics*, JSTOR, p. 649–673, 1987.

LIDL, R.; NIEDERREITER, H. *Finite Fields*. 2. ed. Cambridge, OK: Cambridge University Press, 1997. ISBN 0-512-39231-4.

MENEZES, A. J.; OORSCHOT, P. C. V.; VANSTONE, S. A. *Handbook of applied cryptography*. Florida, USA: CRC press, 1996. ISBN 978-0-8493-8523-0.

MICALI, S.; RABIN, M.; VADHAN, S. Verifiable random functions. In IEEE. *40th Annual Symposium on Foundations of Computer Science*. New York, USA, 1999. p. 120–130.

MILLER, V. Use of elliptic curves in cryptography. In SPRINGER. *Advances in Cryptology (CRYPTO'85)*. Santa Barbara, USA, 1986. p. 417–426.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY, U.S. DEPARTMENT OF COMMERCE. *Recommendation for Key Management – Part 1: General (Revision 3)*. Gaithersburg, USA, 2012. Available from Internet: <http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf>. Cited 11/08/2016.

____. *Special Publication (SP 800-121 Revision 1): Guide to Bluetooth Security*. Gaithersburg, USA, 2012. Available from Internet: <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-121r1.pdf>. Cited 11/08/2016.

____. *Federal Information Processing Standard (FIPS PUB 186-4): Digital Signature Standard (DSS)*. Gaithersburg, USA, 2013. Available from Internet: <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>. Cited 11/08/2016.

____. *Federal Information Processing Standard (FIPS PUB 202): SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Gaithersburg, USA, 2015. Available from Internet: <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>. Cited 11/08/2016.

OKAMOTO, T. Provably secure and practical identification schemes and corresponding signature schemes. In SPRINGER. *Advances in Cryptology (CRYPTO'92)*. Santa Barbara, USA, 1993. p. 31–53.

ONLINE GAMES KINGDOM. *Trading Card Games - Big List of CCGs & TCGs*. —. Available from Internet: <http://tradingcardgames.com/>. Cited 11/08/2016.

PITTMAN, D.; GAUTHIERDICKEY, C. Match+Guardian: a secure peer-to-peer trading card game protocol. *Multimedia systems*, Springer-Verlag, vol. 19, no. 3, p. 303–314, 2013.

POINTCHEVAL, D.; SANDERS, O. Short randomizable signatures. In SPRINGER. *Cryptographers' Track at the RSA Conference*. San Francisco, USA, 2016. p. 111–126.

PRITCHARD, M. How to hurt the hackers: The scoop on internet cheating and how you can combat it. *Gamasutra, July*, vol. 24, 2000. Available from Internet: <http://www.gamasutra.com/view/feature/131557/how_to_hurt_the_hackers_the_scoop_.php>. Cited 11/08/2016.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, ACM, New York, USA, vol. 21, no. 2, p. 120–126, 1978.

ROCA, J. C.; FEIXAS, F. S.; DOMINGO-FERRER, J. *Contributions to mental poker*. PhD Thesis (PhD) — Universitat Autònoma de Barcelona, Bellaterra, Spain, 2006.

SHAMIR, A.; RIVEST, R. L.; ADLEMAN, L. M. Mental poker. In KLARNER, D. A. (Ed.). *The Mathematical Gardner*. Belmont, USA: Wadsworth International, 1981. p. 37–43. ISBN 978-1-4684-6688-1.

SHOUP, V. *A computational introduction to number theory and algebra*. 2. ed. Cambridge, UK: Cambridge University Press, 2008. ISBN 978-0-512-51644-0.

SILVA, M. V. M.; SIMPLICIO JR., M. A. A secure protocol for exchanging cards in p2p trading card games based on transferable e-cash. In SBC. *Proceedings of the XV Brazilian Symposium on Information and Computational Systems Security (SBSeg 2015)*. Florianópolis, Brazil, 2015. p. 184–197. Available from Internet: <http://sbseg2015.univali.br/anais/AnaisSBSeg2015Completo.pdf>. Cited 11/08/2016.

SIMPLICIO JR., M. A. et al. SecureTCG: a lightweight cheating-detection protocol for P2P multiplayer online trading card games. *Security and Communication Networks*, Wiley Online Library, vol. 7, no. 12, p. 2412–2431, 2014.

____. Lightweight and escrow-less authenticated key agreement for the internet of things. *Computer Communications*, Elsevier, 2016. In press.

WEIL, A. L'arithmétique sur les courbes algébriques. *Thèses françaises de l'entre-deux-guerres*, Kluwer Academic Publishers, vol. 95, p. 1–35, 1928.

WIZARDS OF THE COAST. *Magic: the Gathering Tournament Rules*. 2014. Available from Internet: <http://www.wizards.com/contentresources/wizards/wpn/main/documents/magic_the_gathering_tournament_rules_pdf1.pdf>. Cited 11/08/2016.

YAN, J. J.; CHOI, H.-J. Security issues in online games. *The Electronic Library*, MCB UP Ltd, vol. 20, no. 2, p. 125–133, 2002.

APPENDIX A – CONVERSION OF ILV-SIGNATURE PROTOCOL TO ASYMMETRIC PAIRING SETTING

The method proposed in (ABE et al., 2014) for converting a cryptographic protocol from Type 1 (symmetric) pairing to Type 3 (asymmetric) is divided in four steps:

Describe: In this step, the dependency graph for each method is created, associating each group element of the original system (object elements and security problem elements) to the ones used in its computation, and tracing it to group elements that cannot be part of the same source group when both are inputs to the same pairing.

Merge: The dependency graphs are merged into one whole graph, blending the dependencies for the entire system.

Split: The whole dependency graph is split into two graphs, one for each group, respecting the elements that must (or must not) be on the same group, duplicating elements when necessary.

Derivate: Describe the resulting scheme in function on the new elements, considering order in they are input to pairings, and with converted security assumptions with duplicated elements.

As described in Section 3.8, this method is applied on the signature scheme from (IZABACHÈNE; LIBERT; VERGNAUD, 2011) that was originally proposed in the symmetric setting. The summary of the resulting scheme is presented in Section 3.8.1, and here we present the detailed application of the conversion.

Here, we use the original notation from (IZABACHÈNE; LIBERT; VERGNAUD, 2011), that sometimes differs from that of the rest of this document.

A.1 Description of the original protocol

First we need to describe the original protocol from which we will build the dependency graph. This description is necessary because elements that are directly derivated from other elements must belong to the same group, and elements that are used together in some bilinear pairing must belong to different source groups. To build the graph, each pairing is indexed to differentiate its input elements. Since not all methods create new elements, only the ones that do are described here. Namely: KeyGen (Figure 50), Sign (Figure 51), Verify (Figure 52), Commit (Figure 53), WitGen (Figure 54) and VerifyWit (Figure 55).

The remaining methods do not provide new elements or are directly adapted from the previous ones. *UpdateComm* uses the same objects as *Commit*, and *ObtainSig – IssueSig* uses the same as *Commit* and *Sign*. And the proof methods *ProveCom* and *VerifyProofCom* use the same equations as *VerifyWit*, and *ProveSig* and *VerifyProofSig* use the same equation as *VerifyWit* and *Sign*, all together with unconstrained zero knowledge equations.

Besides the methods, the resulting protocol must consider all computational problems considered unsolvable to guarantee its security. As each instance is presented, the reduction of a forgery to one problem is also presented, so that the proof of security results in alternative problems with the corresponding elements. We present the

Figure 50: P-signature (original): Key generation protocol

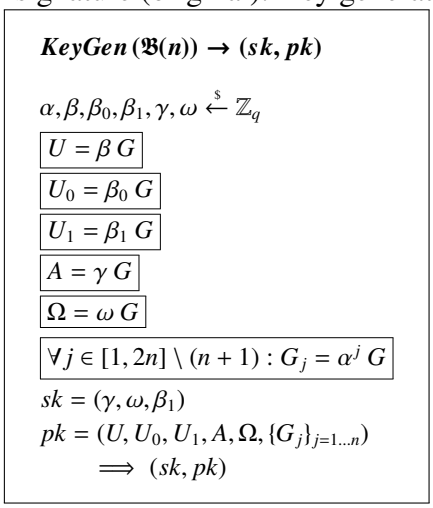


Figure 51: P-signature (original): Sign protocol

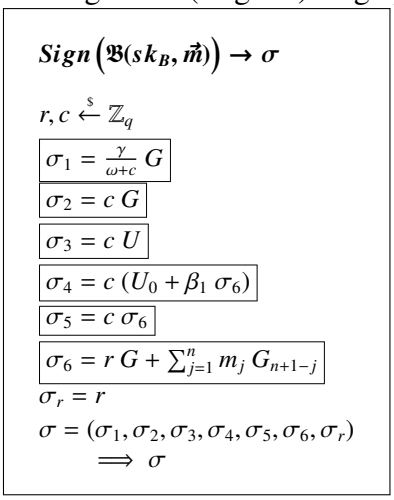
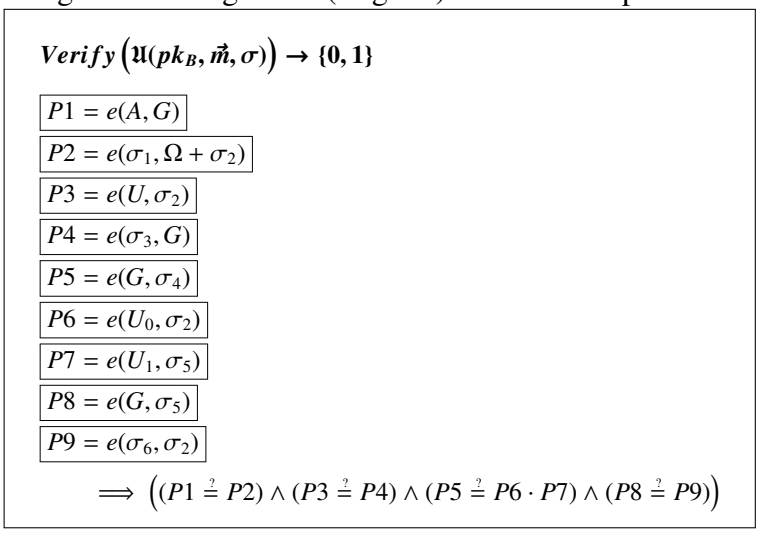


Figure 52: P-signature (original): Verification protocol



instances to the problems HSDH, l-FlexDH and n-FlexDHE in Figures 56, 58 and 60, respectively. The reduction to these instances are presented, respectively, in Figures 57, 59 and 61.

For our construction, only the equality (or pertinence) relation is necessary, so the inequality and inner product relations are disconsidered in our conversion.

With the protocol completely defined, we can construct the dependency graphs. The nodes of these graphs are all elements used or produced from any derivation, and the index of the input of pairing computation (nodes with larger . For each element derivated from another one, a straight arrow indicates that both elements must belong to the same group. The same arrow indicates if an element is used as input of some pairing computation, to relate to which input (the first or the second) of which pairing (all indexed by now) it is used.

The resulting graphs for each protocol described above are presented in Figures 62 (KeyGen), 63 (Sign), 64 (Verify), 65 (Commit), 66 (WitGen) and 67 (VerifyWit).

We also include dependency graphs for each computational problem instance and its respective security reduction. They are described in Figures 68 (HSDH instance), 69 (reduction to HSDH), 70 (FlexDH instance), 71 (reduction to FlexDH), 72 (n-FlexDHE instance) and 73 (reduction to n-FlexDHE).

A.2 Merging the dependency graphs

When all graphs are defined, we need to merge all graphs in one graph that represents all group elements of the protocol. For any pair of node with the same identifier, they become only one note that receives all ancestors and sends all successors from both nodes. The resulting graph is presented in Figure 74. It is important to note that, for this protocol, the commitment C and the sixth element of the signature σ_6 represent the same element. Therefore, they are merged into one node represented by σ_6.

Figure 53: P-signature (original): Commit protocol

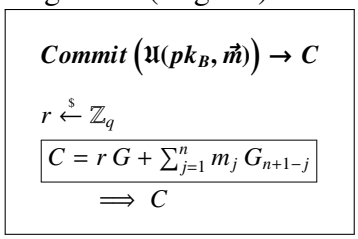


Figure 54: P-signature (original): Witness generation protocol

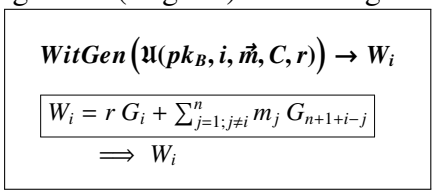


Figure 55: P-signature (original): Witness verification protocol

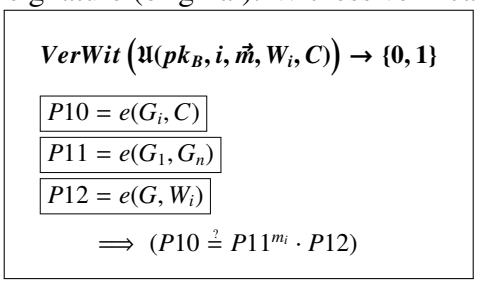


Figure 56: P-signature (assumptions): l-HSDH instance

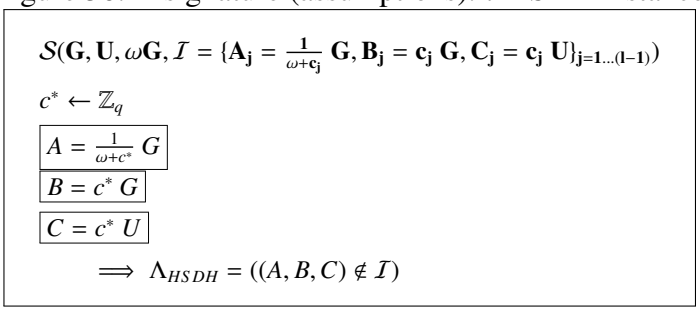


Figure 57: P-signature (assumptions): Security reduction to l-HSDH

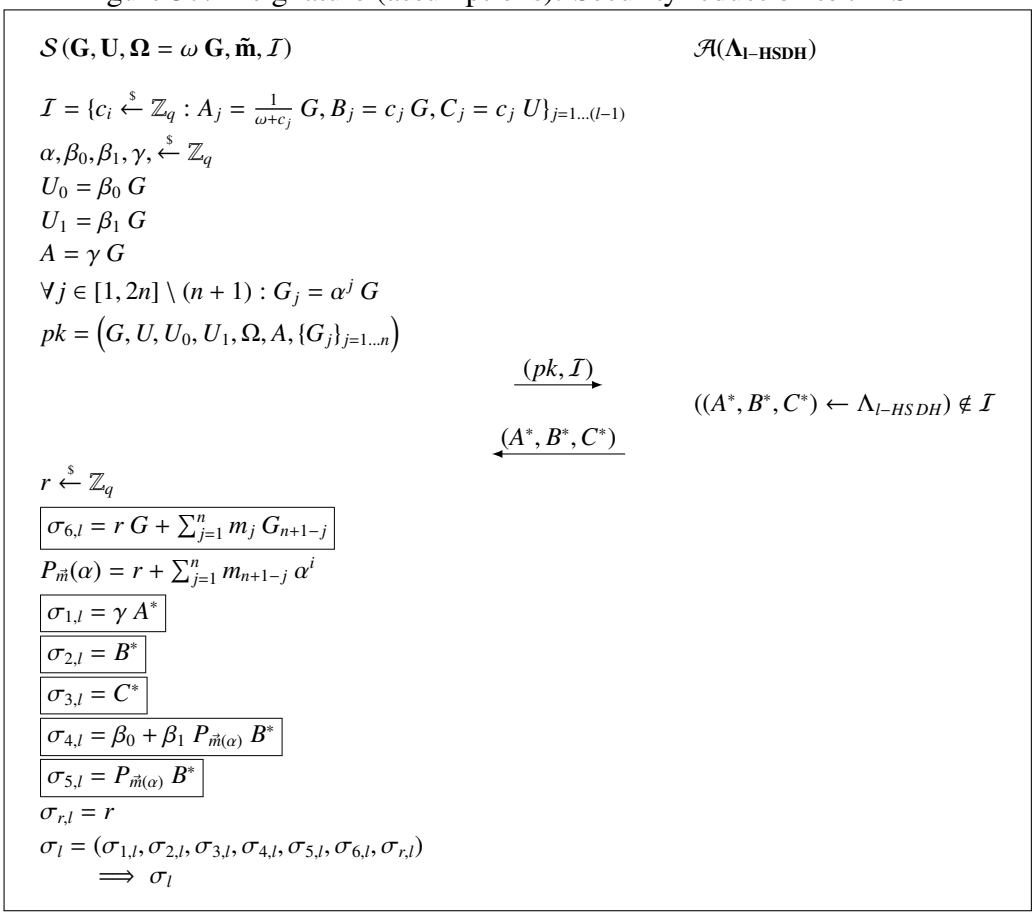


Figure 58: P-signature (assumptions): FlexDH instance

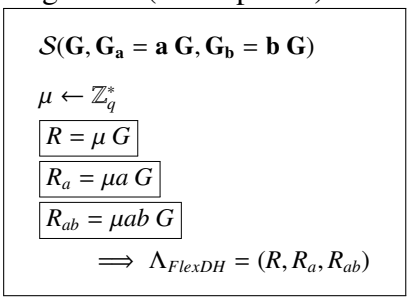


Figure 59: P-signature (assumptions): Security reduction to FlexDH

$\mathcal{S}(\mathbf{G}, \mathbf{G_a} = \mathbf{a}\ \mathbf{G}, \mathbf{G_b} = \mathbf{b}\ \mathbf{G}, \mathbf{U}, \{\tilde{\mathbf{m}}_\mathbf{j}\}_{\mathbf{j=1...l}})$ $\qquad$ $\mathcal{A}(\mathbf{\Lambda_{FlexDH}})$

$\omega, \gamma_a, \gamma_u, \tau_0, \tau_1, \rho_0, \rho_1, \alpha \xleftarrow{\$} \mathbb{Z}_q$
$\Omega = \omega\ G$
$\boxed{A = \gamma_a\ (G_a + \Omega)}$
$U = \gamma_u\ G$
$\boxed{U_0 = \rho_0\ G + \tau_0\ G_b}$
$\boxed{U_1 = \rho_1\ G + \tau_1\ G_b}$
$\forall\, j \in [1, 2n] \setminus (n+1) : G_j = \alpha^j\ G$
$q^* \leftarrow [1, l]$
$pk = \left(G, U, U_0, U_1, \Omega, A, \{G_j\}_{j=1...n}\right)$

For each $k \in [1, l], \vec{m} = \vec{m}_i$
$P_0(\alpha) = \sum_{j=1}^{|m|} m_{|m|+1-j}\ \alpha^j$
If $k \neq q^*$:
$r, c_k \xleftarrow{\$} \mathbb{Z}_q$
$\sigma_6 = r\ G + \sum_{j=1}^{n} m_j\ G_{n+1-j}$
$P_{\vec{m}}(\alpha) = r + P_0(\alpha)$
$\sigma_1 = \frac{\gamma_a}{\omega + c_k}\ (G_a + \Omega)$
$\sigma_2 = c_k\ G$
$\sigma_3 = c_k\ U$
$\sigma_4 = c_k\ ((\rho_0 + P_{\vec{m}}(\alpha)\ \rho_1)\ G + (\tau_0 + P_{\vec{m}}(\alpha)\ \tau_1)\ G_b)$
$\sigma_5 = c_k\ P_{\vec{m}}(\alpha)\ G, \quad \sigma_r = r$
If $k = q^*$:
$r = P_0(\alpha) - \frac{\tau_0}{\tau_1}$
$c_k = c_{q^*} = a$
$P_{\vec{m}}(\alpha) = r + P_0(\alpha)$
$\sigma_6 = r\ G + \sum_{j=1}^{n} m_j\ G_{n+1-j}$
$\sigma_1 = \gamma_a\ G$
$\boxed{\sigma_2 = G_a}$
$\boxed{\sigma_3 = \gamma_u\ G_a}$
$\boxed{\sigma_4 = (\rho_0 + P_{\vec{m}}(\alpha)\ \rho_1)\ G_a}$
$\boxed{\sigma_5 = P_{\vec{m}}(\alpha)\ G_a}$
$\sigma_r = r$
$\sigma_k = (\sigma_{1,k}, \sigma_{2,k}, \sigma_{3,k}, \sigma_{4,k}, \sigma_{5,k}, \sigma_{6,k}, \sigma_{r,k})$

$\xrightarrow{pk, \{\vec{m}_j, \sigma_j\}_{j=1...l}}$

$\sigma^* \xleftarrow{\$} \{\sigma_j\}_{j=1...l}$
If $\sigma^* \neq \sigma_{q^*}$: FAIL
$\sigma_{q^*} \leftarrow \{\sigma_j\}_{j=1...l}, \sigma_{q^*} \neq \sigma^*$ such that $(R, R_a, R_{ab}) \in \Lambda_{FlexDH}$, where:
$\boxed{R = \sigma_6^* - \sigma_{q^*,6}}$
$\boxed{R_a = \sigma_5^* - \sigma_{q^*,5}}$
$\boxed{R_{ab} = \sigma_4^* + \sigma_{q^*,5} - \sigma_{q^*,5} - \sigma_4^*}$
$\sigma^* = (\sigma_1^*, \sigma_2^*, \sigma_3^*, \sigma_{q^*,4}, \sigma_{q^*,5}, \sigma_{q^*,6}, \sigma_{q^*,r})$

$\xleftarrow{\sigma^*}$

$\implies \sigma^*$

Figure 60: P-signature (assumptions): n-FlexDHE instance

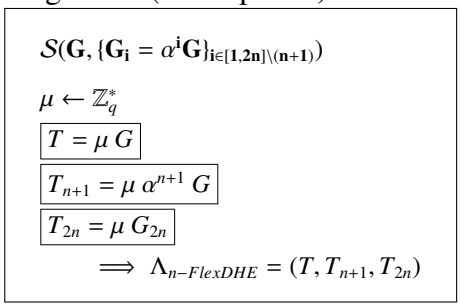


Figure 61: P-signature (assumptions): Security reduction to n-FlexDHE

$\mathcal{S}(\{\mathbf{G_i} = \mathbf{G}^{\alpha^{\mathbf{i}}}\}_{\mathbf{i}\in[\mathbf{1,2n}]\setminus(\mathbf{n+1})})$

$\beta, \beta_0, \beta_1, \gamma, \omega \xleftarrow{\$} \mathbb{Z}_q$
$U = \beta\, G$
$U_0 = \beta_0\, G$
$U_1 = \beta_1\, G$
$A = \gamma\, G$
$\Omega = \omega\, G$
$sk = (\gamma, \omega, \beta_1)$
$pk = (G, U, U_0, U_1, \Omega, A, \{G_i\}_{i\in[1,2n]\setminus(n+1)})$
$\vec{m} \leftarrow \mathbb{Z}_q^n$
$\sigma = Sign(sk, \vec{m})$

(pk, σ)

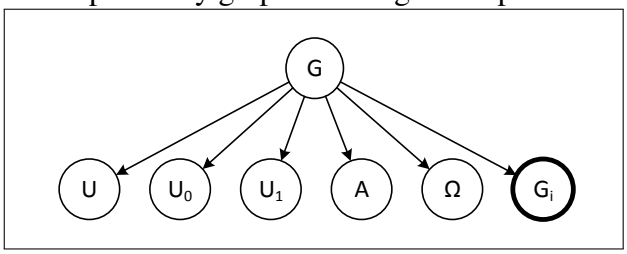
$\mathcal{A}(\mathbf{\Lambda_{n-FlexDHE}})$

$\vec{x^*} \leftarrow \mathbb{Z}_q^n \implies \exists i \in [1, n] : x_i^* \neq m_i$
$F_{i,1} = x_i^*\, G_1$
$F_{i,2} = x_i^*\, G$
$F_{i,3} = x_i^*\, G_{2n}$
$F(x_i^*) = (F_{i,1}, F_{i,2}, F_{i,3})$
$P13 = e(G_i, \sigma_6)$
$P14 = e(G, W^*)$
$P15 = e(F_{i,1}, G_n)$
$W^* \leftarrow \mathbb{G} : P13 = P14 \cdot P15$ such that $(T, T_{n+1}, T_{2n}) \in \Lambda_{n-FlexDHE}$, where:
$T = m_i\, G - F_{i,2}$
$T_{n+1} = W^* - W'$
$T_{2n} = m_i\, G_{2n} - F_{i,3}$
$Pred^* = (W^*, i, F(x_i^*))$

$Pred^*$

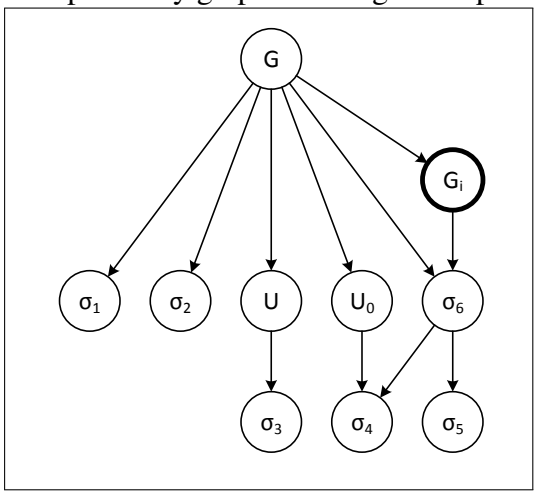
$\implies (\sigma, Pred^*)$

Figure 62: Dependency graph for P-signature protocol KeyGen



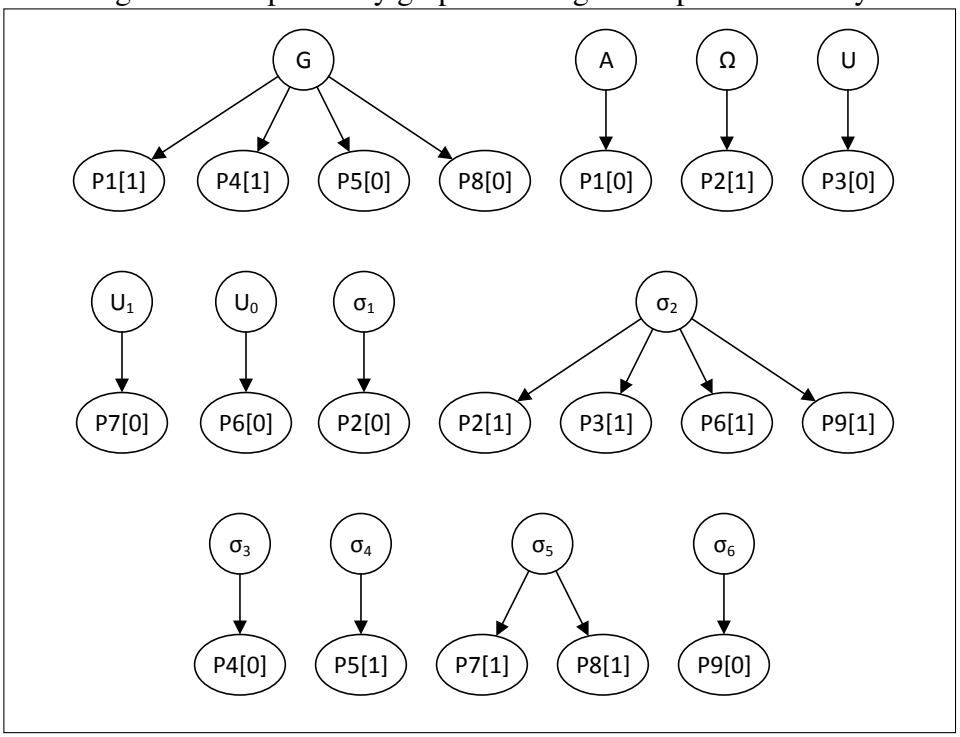
Source: Author

Figure 63: Dependency graph for P-signature protocol Sign



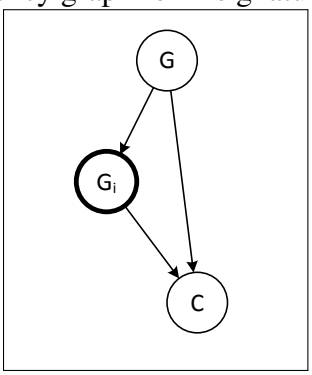
Source: Author

Figure 64: Dependency graph for P-signature protocol Verify



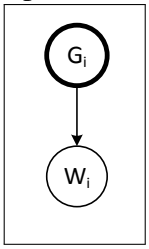
Source: Author

Figure 65: Dependency graph for P-signature protocol Commit



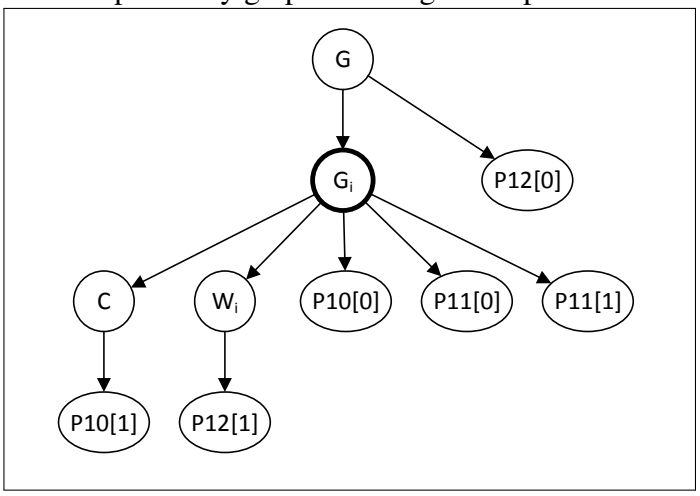
Source: Author

Figure 66: Dependency graph for P-signature protocol WitGen



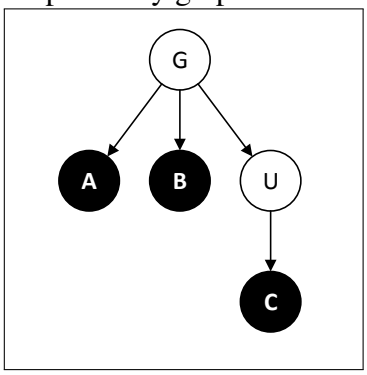
Source: Author

Figure 67: Dependency graph for P-signature protocol VerifyWit



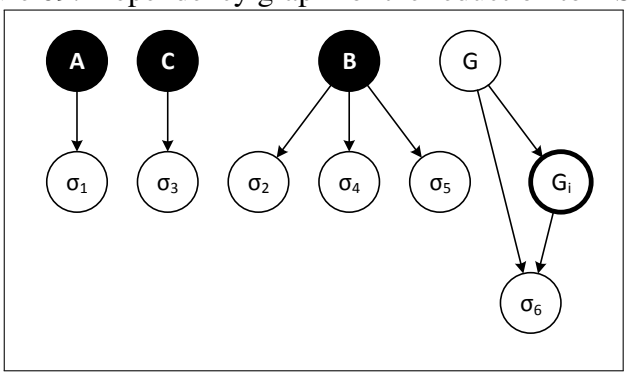
Source: Author

Figure 68: Dependency graph for HSDH instance



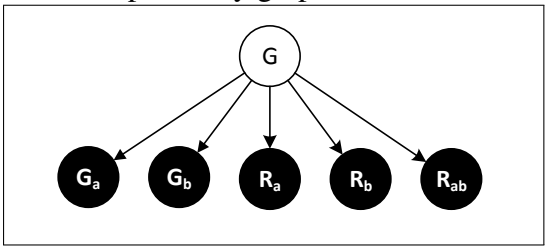
Source: Author

Figure 69: Dependency graph for the reduction to HSDH



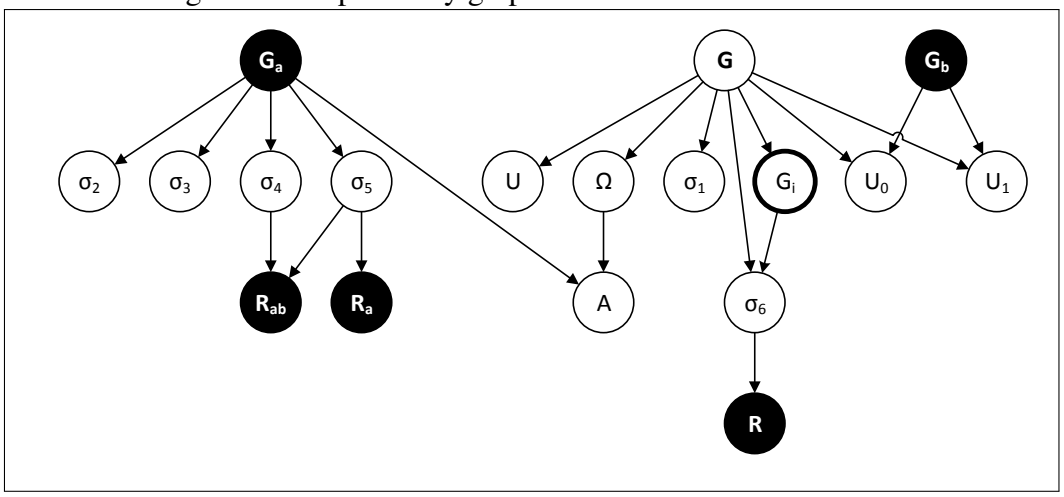
Source: Author

Figure 70: Dependency graph for FlexDH instance



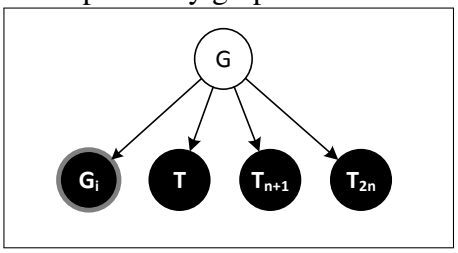
Source: Author

Figure 71: Dependency graph for the reduction to FlexDH



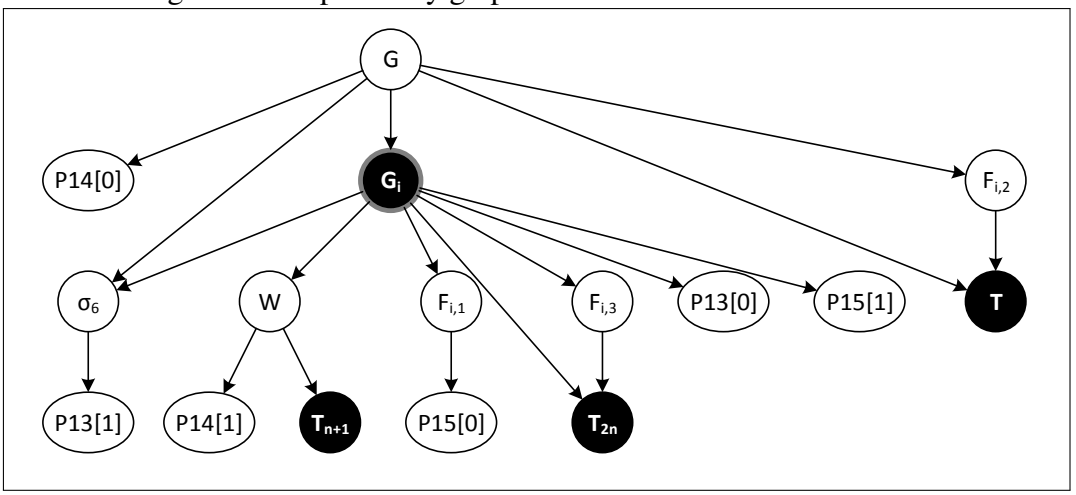
Source: Author

Figure 72: Dependency graph n-FlexDHE instance



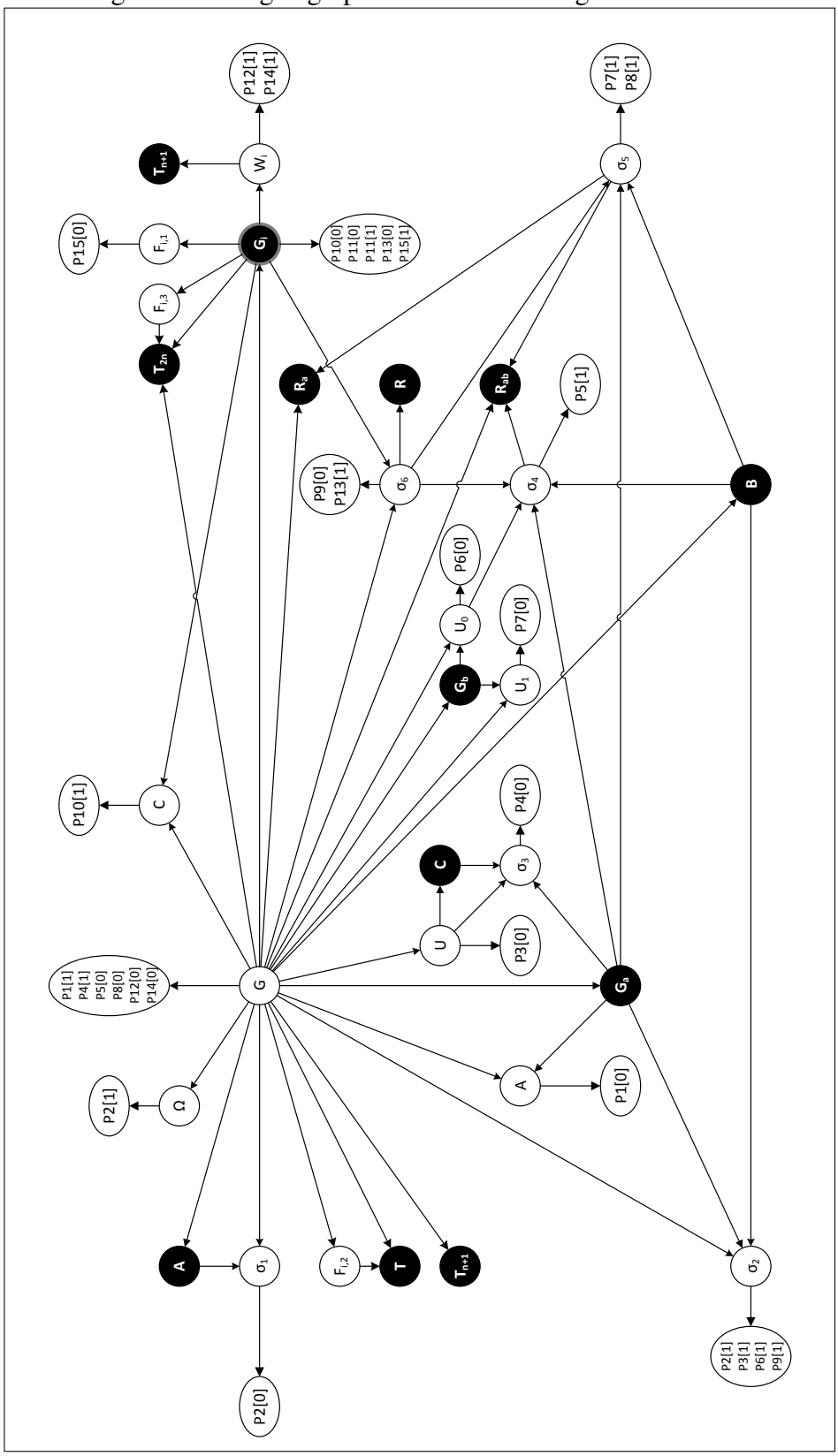
Source: Author

Figure 73: Dependency graph for the reduction to n-FlexDHE



Source: Author

Figure 74: Merged graph for the entire P-signature scheme



Source: Author

A.3 Splitting the graph

With the entire graph structure, it becomes necessary to split the graph into two parts, one for each source group of the pairing. First we begin by choosing a strategy to choose with partition to use (since there are 2^{n_P} possible partitions, although not all of them are valid ones).

For this application, a valid strategy is *to reduce storage and communication*. Elements of $\mathbb{G}_2$ are usually bigger than elements of $\mathbb{G}_1$, and the operations among them are more cumbersome. To achieve the splitting objective, we will try to minimize the number of elements in $\mathbb{G}_2$.

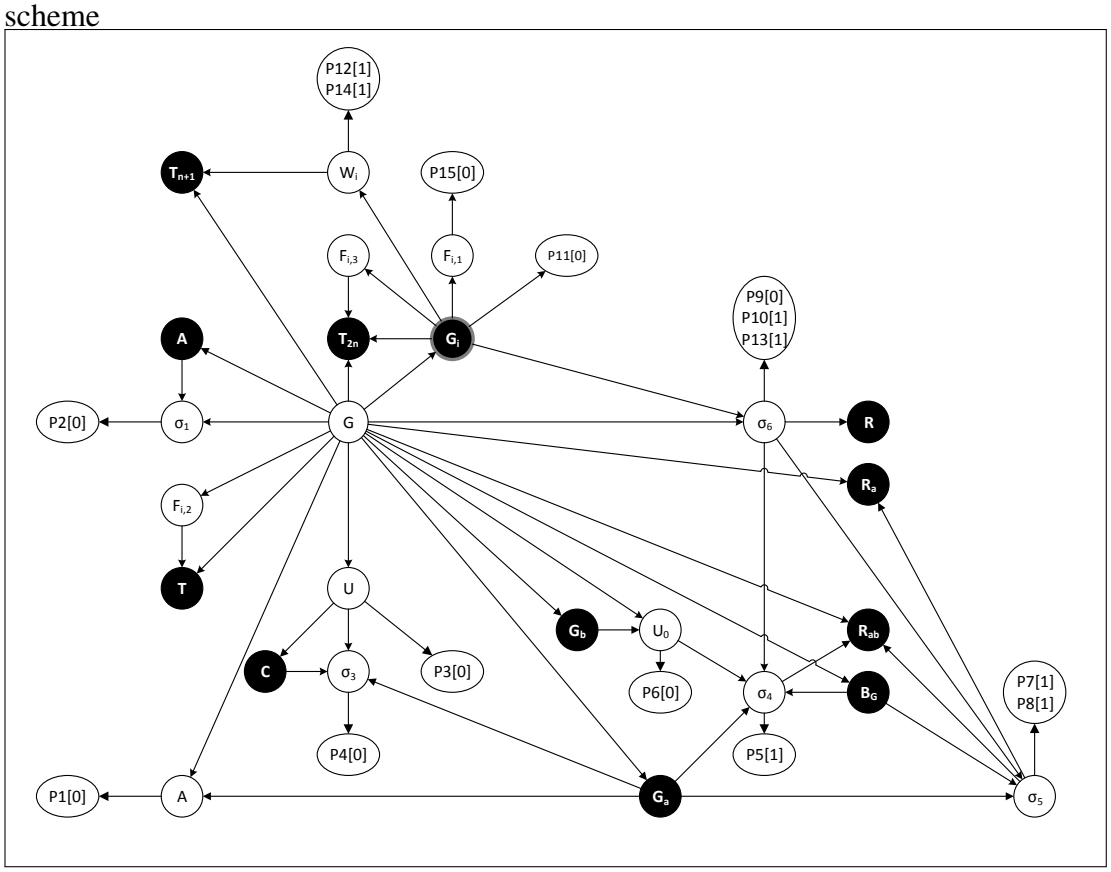
For each described pairing, one source element belongs to $\mathbb{G}_1$ and the other to $\mathbb{G}_2$. Whenever possible, we maintain "volatile" elements (i.e., signature elements, commitments) in $\mathbb{G}_1$, if the "static" elements (i.e., generators, public key elements) can belong to $\mathbb{G}_1$. If some element is required by both pairing source groups, then it is duplicated and each node belongs exclusively to one group. Even though it reduces the number of group elements to represent each protocol element, the proof of knowledge protocol uses commitments to variables and constants with the same size to both pairing source groups. Thereby this strategy does not reduces the proof's load.

By analyzing the graph, it is possible to select the witnesses W_i and almost all signature elements (except for σ_2) to $\mathbb{G}_1$. The resulting graphs as presented in Figure 75 and 76, for elements that belongs to $\mathbb{G}_1$ and to $\mathbb{G}_2$ respectively.

To simplify viewing the splitting results, Table 11 presents the elements that belong to each group, including the duplicated elements.

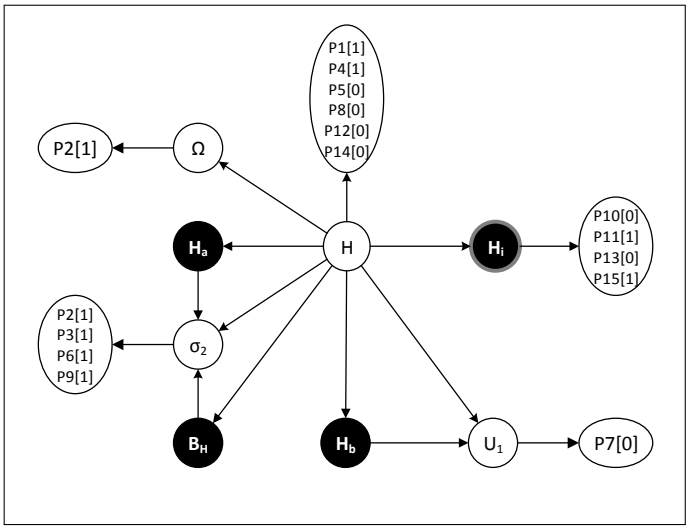
We also note that this partition is likely different from the one presented in (SILVA; SIMPLICIO JR., 2015) because different strategies were used to find them. While here we intend to reduce the number of elements in $\mathbb{G}_2$, the partition chosen in (SILVA;

Figure 75: Split dependency graph for elements in $\mathbb{G}_1$ for the converted P-signature scheme



Source: Author

Figure 76: Split dependency graph for elements in $\mathbb{G}_1$ for the converted P-signature scheme



Source: Author

Table 11: Elements of each source group in the converted P-signature scheme after splitting the pairing groups

Protocol element	$\mathbb{G}_1$	$\mathbb{G}_2$	$(\mathbb{G}_1, \mathbb{G}_2)$
Generator			(G, H)
Public Key	U, U_0, A	U_1, Ω	(G_i, H_i)
Signature	$\sigma_1, \sigma_3, \sigma_4, \sigma_5, \sigma_6$	σ_2	
Witness	W_i		
q-HSDH Instance	A, C		(B_G, B_H)
FlexDH Instance	R_a, R_b, R_{ab}		$(G_a, H_a), (G_b, H_b)$
n-FlexDHE Instance	$T, T_{n+1}, T_{2n}, F_{i,1}, F_{i,2}, F_{i,3}$		(G_i, H_i)

Source: Author

SIMPLICIO JR., 2015) was found by a greedy algorithm.

A.4 Derivating the protocol

After defining to which group each protocol element must belong, it is just necessary to rewrite the entire protocol so that it is set in the asymmetric setting $\Lambda = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, G, H, e)$. The following protocols are presented: Namely: KeyGen (Figure 77), Sign (Figure 78), Verify (Figure 79), Commit (Figure 80), WitGen (Figure 81) and VerifyWit (Figure 82).

Besides updating the protocols, we also have to update the computational problems assumed unsolvable in viable time. The new problems were defined in Section 3.5.

Figure 77: P-signature (converted): Key generation protocol

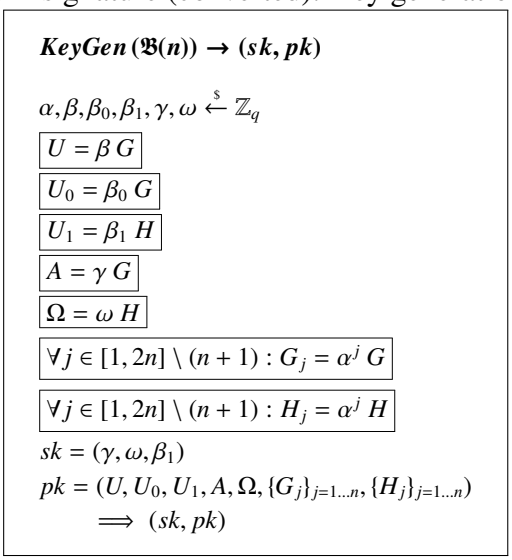


Figure 78: P-signature (converted): Sign protocol

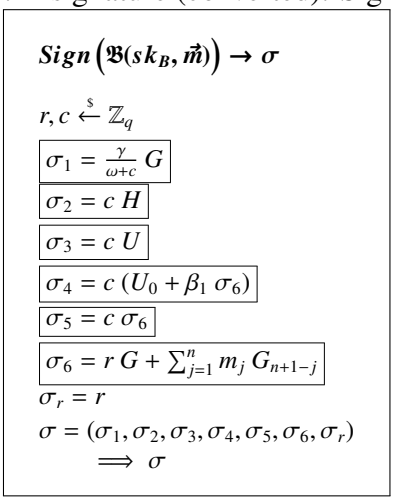


Figure 79: P-signature (converted): Verification protocol

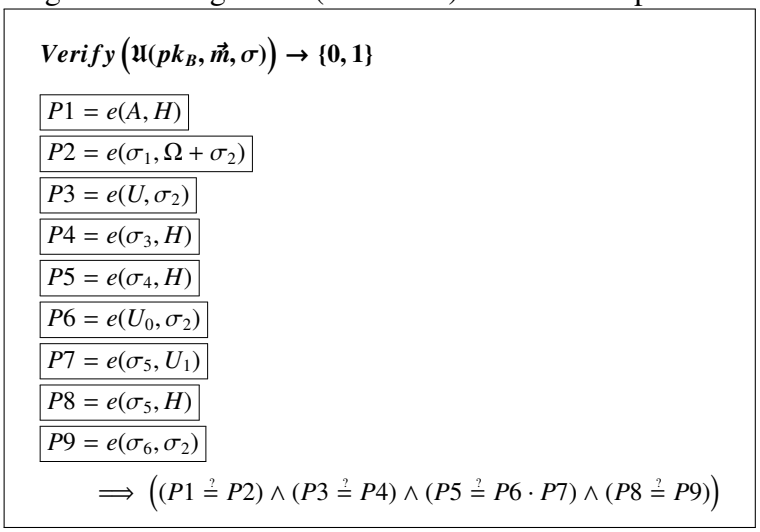


Figure 80: P-signature (converted): Commit protocol

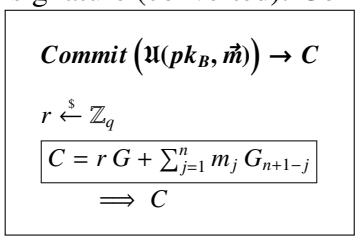


Figure 81: P-signature (converted): Witness generation protocol

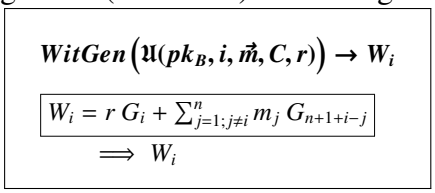
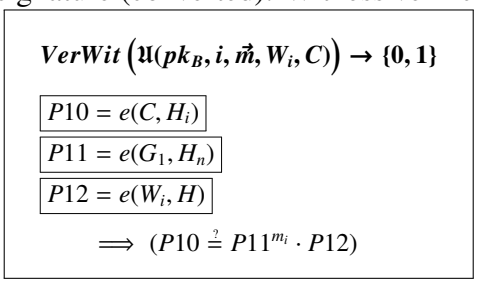


Figure 82: P-signature (converted): Witness verification protocol



APPENDIX B – RELIC PARAMETERS

We now summarize the relevant options for compiling the RELiC toolkit. The 64-bit architecture parameters are presented in Table 12. The methods for operations with large integers are presented in Table 13. The groups for the pairing computation and elliptic curves are presented in Table 14. And the miscellaneous algorithms are presented in Table 15.

This results in the use of the SHA-256 hash function, the underlying operating system's pseudorandom number generator and the use of the curve $EC/\mathbb{F}_n : y^2 = x^3 + 2$ with the following parameters group and generator coordinates:

n	997569478656000733506969600000219732480000000031824000000000019
$G.x$	$0x2523648240000001BA344D8000000008612100000000013A700000000000012$
$G.y$	1

Table 12: Compiling parameters for the RELiC library (architecture)

ALLOC=	AUTO	Automatic memory allocation (dynamically on demand)
ARCH=	X64	64-bit architecture
ARITH=	x64-asm-254	Use architecture specific assembly code
WORD=	64	Operations in words of 64 bits

Source: Author

Table 13: Compiling parameters for the RELiC library (big numbers)

BN_MAGNI=	DOUBLE	Double precision integers store twice as many words
BN_METHD=	COMBA;	Comba multiplication
	COMBA;	Comba squaring
	MONTY;	Montgomery modular reduction
	SLIDE;	Sliding windows modular exponentiation
	BASIC;	Euclid's sieve GCD algorithm
	BASIC	Standard prime generation
BN_PRECI=	1024	Use big numbers of up to 1024 bits
FP-METHD=	INTEG;	Integrated modular addition
	INTEG;	Integrated modular multiplication
	INTEG;	Integrated modular squaring
	MONTY;	Montgomery modular reduction
	LOWER;	Inversion to lower level
	SLIDE	Sliding windows exponentiation
FP_PRIME=	254	Use finite field of 254-bit integers
FP_WIDTH=	4	Window width for exponentiation method

Source: Author

Table 14: Compiling parameters for the RELiC library (elliptic curve and pairings)

EC=METHD=	PRIME	Use prime elliptic curve
EP=DEPTH=	4	Depth of precomputation table
EP_METHD=	PROJC;	Jacobin projective coordinates
	LWNAF;	Left-to-right windows NAF method
	COMBS;	Single-table Comb method for point multiplication
	INTER	Interleaving of windows NAF for simultaneous scalar multiplication
EP_MIXED=	ON	Use mixed coordinates (affine and Jacobian)
EP_PLAIN=	ON	Ordinary elliptic curve (without endomorphism)
EP_PRECO=	ON	Use precomputed table for curve generator
FPX_METHD=	INTEG;	Quadratic extension field with embedded modular reduction
	INTEG;	Cubic extension field with embedded modular reduction
	LAZYR	Lazy-reduced extension field arithmetic
PP_METHD=	LAZYR;	Lazy-reduced extension field
	OATEP	Use optimal ate pairing

Source: Author

Table 15: Compiling parameters for the RELiC library (hash and pseudorandom generator)

MD_METHD=	SH256	Using SHA-256 hash function
RAND=	UDEV	Unix udev blocking pseudorandom generator

Source: Author