A Secure Protocol for Exchanging Cards in P2P Trading Card Games

Ved Danait(22B1818) and Geet Singhi(22B1035)

May 2024

1 Abstract

Trading card games (TCG) distinguish from traditional card games mainly because the cards are not shared between players in a match. Instead, users play with the cards they own (e.g., purchased or traded with other players), which corresponds 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 without such entity remains a challenging task. Potential solutions are related to e-cash protocols, but, unlike the latter, TCGs require users to play with the cards under their possession, not only to be able to pass those cards over. We discuss the security requirements of TCGs and how they relate to e-cash. We then propose a concrete, TTP-free protocol for anonymously trading cards, using as basis a secure transferable e-cash protocol.

2 Introduction

The proposed scheme is based on existing transferable e-cash protocols, with the required adaptations for allowing players to do the following :

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. Verify the validity of the card without the intervention of a third party, independently of the number of previous owners the card has ever had
4. 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

3 Architecture

The game server is responsible for any action that requires a trusted authority or centralized information storage. One of its primary roles is to serve as a *registration center* for players: to enroll in the system, a user must register with a unique identifier (e.g. 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*, 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 cards available in the game, as new releases usually add several new cards to the game. Finally, the server is also the entity that plays the role of *game auditor*, verifying claims regarding cheating attempts and eventually punishing those responsible for misbehavior.For example, the players may send after-match information to the server to prove that a user cheated, e.g., by modifying the sequence or contents of their deck during a match. 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 traded it earlier).

4 Representation of Cards

The minimal representation of a card C in a typical TCG (Trading Card Game) corresponds to a tuple C = (ID, d, V, owner), where: ID is the card's unique identifier; d is the card's game-specific information, which defines how it affects the game, which are the conditions for it to be played, etc.; V is some validation information, which allows any player to verify that the card was indeed issued by the game provider; and owner is the information that allows the card's current owner to be identified.

5 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, 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 one. 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.

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, or needs to be stored somewhere else to prevent such growth (e.g., in a receipt in an e-cash scheme). To avoid indefinite growth, players may *refresh* their cards, which is equivalent to deposit a coin and get a new, mint version of it.
In transferable e-cash schemes, there is a fundamental problem (called the double-spending problem) where the user might try to spend the same coin twice. In secure transferable e-cash schemes, the server revokes the anonymity of the user. In the context of TCGs, however, the double spending problem is more complicated because players may not only trade, but also use their cards 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, 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.
In summary, five types of cheating can appear when cards are traded: (1) Double-refresh: refreshing a 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 C to obtain a mint version of it, C', but then trading copies of C with other users; and (5) Refresh-then-play: refreshing a card C to obtain a mint version of it, C', but then using C in matches with other players.

6 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-

1. Verifiable stamping: The card market must stamp cards, so their validity and ownership can be verified without the need of contacting the central server.

2. TTP-free transferability: Players should be able to trade cards with each other without the intervention of a TTP (Trusted Third Party), and the

new ownership can also be verified without the need of contacting a trusted server.

3. Anonymity: Suppose that U0 purchases a given card C, and then that card is repeatedly traded among a set of users U1...n before the last owner, Un+1, informs the server about this ownership. In this case, the server only learns the identity of Un+1, while the C's previous owners remain anonymous. In addition, during this process user Ui only learns the identity of Ui-1 and Ui+1.

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

5. 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.

6. 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 identifying users who have duplicated a card (either for trading or playing with it).

6.1 Functions used as building blocks

There are certain functions that are used as building blocks in our proposed scheme.

- PSetup(k) $\rightarrow$ pparams: Generates the set of public parameters pparams for the signature, which corresponds to the parameters of Groth-Sahai proofs under an asymmetric pairing setting. For the sake of simplicity, these parameters are omitted in the descriptions of the remainder functions.
- PKeyGen(n) $\rightarrow$ (pk, sk): Compute the public key and the private key according to the PBC scheme.
- PSign(sk,$\vec{m}$) $\rightarrow \sigma$: Compute the signature for the secret key and the message.
- PVerifySig(sk,σ,$\vec{m}$) $\rightarrow \{0,1\}$: Verify the signature.
- PCommit(pk,$\vec{m}$) $\rightarrow (K,r)$: Compute the commitment.
- PUpdateComm(pk,$\vec{m}$, K)$\rightarrow$ K': Compute and output the new commit.
- PWitGen (pk, i, $\vec{m}$,K,r) $\rightarrow W_i$: If K is a commitment to $\vec{m}$ then compute the witness based on the opening r.
- PVerifyWit(pk,i,m_i,W_i,K) $\rightarrow \{0,1\}$: Returns 1 if and only if the witness is valid.

- PProveCom(pk,$\vec{m}$, K,r) $\rightarrow \phi_K$: Generates witnesses for every message committed and generates a Groth-Sahai proof of knowledge of them.
- PVerifyProofCom(ϕ_K) $\rightarrow \{0,1\}$: Verifies that the Groth-Sahai proof of knowledge ϕ_K was correctly constructed.
- (PObtainSig(pk,$\vec{m}_P$) $\leftrightarrow$ PIssueSig(sk,$\vec{m}_S$) $\rightarrow \sigma$: This is a protocol consisting of the following steps -
    1. The user commits the message $\vec{m}_P$ as $(K, r) =$ PCommit(pk,$\vec{m}_P$) and sends K to the signer with a proof of knowledge $\phi_K =$ PProveCom(pk,$\vec{m}_P$,K,r') that the commitment is valid.
    2. The signer verifies the proof with a call to PVerifyProofCom(ϕ_K), updates the commitment to $K' =$ PUpdateCom(pk,$\vec{m}_S$, K) and blindly signs the commitment with randomly generated seeds and then the User acknowledges the verification.
- PProveSig(pk, $\vec{m}$, σ) $\rightarrow \phi_\sigma$: Generates witnesses for every message signed and generate a Groth-Sahai proof of knowledge for the signature validation, commitment validation and message persistence validation equations.
- PVerifyProofSig(ϕ_σ) $\rightarrow \{0,1\}$: Verifies if the proof of knowledge ϕ_σ was correctly constructed.

7 Relation to compact e-cash

Previous e-cash schemes allowed users to withdraw several coins (i.e. a wallet) within a single message. In the context of TCGs, this scheme is interesting for several reasons :

1. Allows several seed parameters instead of coins to be signed altogether
2. Provides a direct method to identify cheaters, who have their public key recovered, so that the server need not scan the entire database to identify them.

Our proposed solution is based on a modification of an older research paper by S. Canard, A. Gouget, and J. Traore, which achieves transferrability with strong anonymity using the following funtions:

1. Setup() : Bank generates a public-private key pair and publishes its public key along with system parameters.
2. Register() : User generates a public-private key pair based on the system parameters and retrieves a certificate from the bank of the validity of this key pair. The user identity along with the corresponding public key is stored by the bank, which allows the user to be identified in case of double spending.

3. Withdraw() : The user produces seed values and commits them to the bank, which blindly signs these values. This creates a new anonymous wallet with as many coins as the number of seeds provided.

4. Spend() : Users may exchange either unspent coins or previously received coins from their wallets. 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 hold value.

5. 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).

6. Identify() : In case of double spending the bank retrieves the public key of the perpetrator, and applies appropriate penalties.

A wallet W is composed of a private key *pk* of the owner, a serial seed s, a transfer seed t, and a signature σ on these values. A coin C is identified by a serial number S, proof of validity ϕ_S, proof of knowledge on the signature of the wallet (that it currently belongs to) ϕ_σ and a set π_T of j transferences. Each transference is composed of a transference tag T_j and its proof of validity ϕ_{T_j}, a tag of ownership r_j and some public information i_j.
The serial number S is picked at random to provide a unique identifier for each coin. It is then employed in a Verifiable Random Function (f_S) (*serial number generating function*) according to the following equation -

$$f_S(\mathrm{sk}_U, s) = G^{\frac{1}{s+\mathrm{sk}_U}}$$

In this equation s is a serial seed signed in the wallet and sk_U is the private key of the owner (or the index of the coin if more than one coin can be withdrawn) The transference tag is used to identify transferences and is employed in a modified VRF (*transference tag generating function*) according to the following equation -

$$f_T(\mathrm{sk}_U, t, R) = (G^R)^{\mathrm{sk}_U} G^{\frac{1}{t+\mathrm{sk}_U}}$$

Here t is the transference seed signed in the wallet or referenced by the previous transference, sk_U is the secret key of the owner and R is the hash of the private (that contains the owner) and the public (e.g. a timestamp) information of the transference.
Finally, the ownership tag r is a randomly picked seed used to hide the private key of the coin's owner. Similar to S, it is employed in a VRF (*ownership tag generating function*) according to the following equation -

$$f_r(\mathrm{sk}_U, i) = G^{\frac{1}{i+\mathrm{sk}_U}}$$

Here sk_U is the private key of the owner and i is the public information 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.

8 Protocol For Trading Cards

We can now use the building blocks defined in the previous section to create a protocol for Trading Cards. The roles of the registration center $\mathcal{C}$, card market $\mathcal{M}$, $\mathcal{A}$ are played by the game server $\mathcal{G} = \mathcal{C} \cup \mathcal{M} \cup \mathcal{A}$. A card C is represented as C = (ID, d, V, $owner$) where ID is an element of G_1, d an element of Z_q is an encoding of the card, V = (ϕ_{ID}, ϕ_σ) where the two members are proofs of knowledge of construction of the ID and of the signature from the market, and owner = π_T = { T_j, ϕ_{T_j}, r_j, i_j} that corresponds to the records of all owners of the cards, so that for each index j in π_T, T_j is transference tag with PoK ϕ_{T_j}, r_j is the ownership tag and i_j is the public information regarding the transference. Now we put forth the operations involved in the proposed scheme :

1. Setup() : The game server generates the system parameters for two signature schemes to register new players and to stamp new cards. Both of these contain parameters of a Groth-Sahai proof system defined over an asymmetric pairing .The game server also generates two key pairs using *PKeyGen()* to register players and to stamp cards. It then publishes the system parameters and the public keys.

2. Register() : The function takes player ID as input and generates a secret key for the player. It then computes the public key using the secret key and the pairing : $pk_P = e(G, H)^{sk_P}$. The player then computes a proof of knowledge ϕ_P using the secret key. The triple (id_P, pk_P, ϕ_P) is sent to the registration centre $\mathcal{C}$. If the proof ϕ_P is valid then C generates a signature $\sigma_P = PSign(sk_C, \{id_P, pk_P\})$. $\mathcal{P}$ can then present σ_P as his/her certificate.

3. Stamp() : Player can purchase an instance of the card with description d. The player generates a partial identifier seed s' and transference seed t, and the card market $\mathcal{M}$ generates the card's partial identifier component s". Both parties execute an interactive protocol to obtain blind signature using the *PObtainSig* and *PIssueSig* functions that is returned to P together with s". The player then generates a proof of knowledge ϕ_d using *PProveSig*. Subsequently, the player chooses some unique public information like a timestamp and then computes $r_0 = G^{\frac{1}{sk_P + \mathcal{H}(i_0)}}$ and $R_0 = \mathcal{H}(r_0, i_0)$. Player then generates unique identifier ID using a VRF as described in the previous section, a transference tag T_0 using VRF as well, together with proofs of knowledge for both of them associated with

the commitments in the proof of signature ϕ_σ. Finally the player stores the card C,

4. Send() : The receiver chooses some public information as before and then computes r and a proof of validity ϕ_r. Receiver then sends the tuple to the current card holder who parses the card and verifies the proof of validity. If everything is is correct then P1 appropriately generates a new transference tag as well as a proof of knowledge for the construction. The modified card, incorporating the transference tag is then sent to the other player who upon receiving it verifies the construction of the unique identifier ID and the tags $\{T_j\}_{j=0\ldots h}$ as well as if the proof of ownership is valid om respect to the public key pk_{P_2}. If all Proofs are correct the receiver stores the card as its own.

5. Play() : This is implemented in a similar manner as the send function with the player updating the card and generating a new transference tag and a 2 proof of knowledge for the construction to prove that the card was correctly prepared. The receiver verifies these proofs of knowledge and if they are all valid, stores the card locally so that its information can be reported to the game server later and uses the unique identifier to identify this card during the match.

6. Report() : A player sends a card that its opponent has used during the match to the game auditor A. Auditor stores the card and verifies if any card with the same identifier has already been reported. A uses the $Identify()$ function to retrieve the public keys of users who illegally duplicated this card.

7. Refresh() : The player similarly updates the card based on public information and generates new transference tag and PoK of the construction. The updated card is then sent to the game server $\mathcal{G}$. The auditor then stores the card and verifies if there is any other card with the same identifier already reported in RS. For each card the auditor executes the $identify()$ function retrieving the list of cheating users. If no transgressor was returned, both parties execute *Stamp*() to produce a fresh card to the player.

8. Identify() : Game Auditor parses through cards C and C' with the same value of ID. It searches for the first index l where the transference tags do not match and retrieves the public key of the perpetrator. If the index l is larger than the number of hops for any card, this card has already been reported but not been duplicated so the output is empty.

The requirements of a secure card trading system, are fulfilled by the underlying e-cash scheme. The signature on stamping method guarantees verifiability ("own" property), anonymity when stamping (the signer cannot link signatures to new cards), and balance (if the signature is unforgeable, a new card cannot be inconspicuously created without authorization by the card market). The proof of knowledge provides transferability on trading and ad-hoc playing, given its non-interactivity property. It also keeps anonymity when trading, since it is witness-indistinguishable together with the VRF. Finally, the identification method of the e-cash scheme guarantees balance, cheat detection and exculpability.

9 Efficiency Analysis

Signing same n=4 messages with the aforementioned P-signature scheme, a signature proof requires 20 elements in G_1 and 42 in $G-2$. For serial number generation proof we need 24 and 26 elements in G_1 and G_2 respectively. For transference tag generation proof we need 36 and 38 elements in G_1 and G_2 respectively. Considering the composition of the card, for a total of t transferences a card will need 45+38t elements in G_1 and 68 + 38t elements in G_2. The execution time is likely dominated by the pairing computations. Using a GS proof of knowledge, each trade or play of a card corresponds to roughly 150 underlying pairing computations, since each pairing roughly takes about 1ms to run the total time would be around 150ms for each card traded or played. We note that, while these timings are quite reasonable for trading, they may be somewhat cumbersome when playing with a deck having 50 cards since the verification of the deck would take around 7.5 minutes. However this cost can be amortized by preparing the deck beforehand, much before the match starts and the verification of the corresponding Proofs of Knowledge can happen in background during the match which usually takes several minutes. Therefore in practice these costs can be made to be transparent to players.

10 References

A large part of this scheme was based on the e-cash protocol laid forward by J. Camenisch, S. Hohenberger, and A. Lysyanskaya, "Compact e-cash," in Advances in Cryptology (Eurocrypt'05). Springer, 2005, pp. 302321 and further refined by M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya, "Compact e-cash and simulatable vrfs revisited," in Pairing'09. Springer, 2009, pp. 114–131.