On The Centrality of Off-Line E-Cash to Concrete Partial Information Games

Seung Geol Choi[1], Dana Dachman-Soled[2], and Moti Yung[3]

[1] University of Maryland sgchoi@cs.umd.edu
[2] Microsoft Research New England dadachma@microsoft.com
[3] Google Inc. & Columbia University moti@cs.columbia.edu

Abstract. Cryptography has developed numerous protocols for solving "partial information games" that are seemingly paradoxical. Some protocols are generic (e.g., secure multi-party computation) and others, due to the importance of the scenario they represent, are designed to solve a concrete problem directly. Designing efficient and secure protocols for (off-line) e-cash, e-voting, and e-auction are some of the most heavily researched concrete problems, representing various settings where privacy and correctness of the procedure is highly important.

In this work, we initiate the exploration of the relationships among e-cash, e-voting and e-auction in the universal composability (UC) framework, by considering general variants of the three problems. In particular, we first define ideal functionalities for e-cash, e-voting, and e-auction, and then give a construction of a protocol that UC-realizes the e-voting (resp., e-auction) functionality in the e-cash hybrid model. This (black-box) reducibility demonstrates the centrality of off-line e-cash and implies that designing a solution to e-cash may bear fruits in other areas. Constructing a solution to one protocol problem based on a second protocol problem has been traditional in cryptography, but typically has concentrated on building complex protocols on simple primitives (e.g., secure multi-party computation from Oblivious Transfer, signature from one-way functions, etc.). The novelty here is reducibility among mature protocols and using the ideal functionality as a design tool in realizing other ideal functionalities. We suggest this new approach, and we only consider the very basic general properties from the various primitives to demonstrate its viability. Namely, we only consider the basic coin e-cash model, the e-voting that is correct and private and relies on trusted registration, and e-auction relying on a trusted auctioneer. Naturally, relationships among protocols with further properties (i.e., extended functionalities), using the approach advocated herein, are left as open questions.

1 Introduction

1.1 Motivation

Research on the security and privacy of cryptographic protocols where parties share information (i.e., partial information games) is a major area of research in cryptography. Many scenarios which seem paradoxical and unsolvable have been shown to be realizable, based on the power of information distribution and/or that of public-key cryptography.

While a good deal of research has been performed on constructing generic protocols (i.e., general secure multi-party computation) [Yao86,GMW87], there still exist important real life procedures that deserve special considerations: e-cash [Cha83,CFN88], e-voting [Cha81,CF85,BY86] and e-auction [FR96,NPS99]. Due to the great impact on the viability of cyberspace transactions that the successful deployment of these protocols brings about, they have attracted numerous researchers, and for decades, much work has been done on defining security and constructing secure protocols for these tasks. These specific partial information games share some basic configuration: Each of these games is structured so that players are either authorities (i.e., banks, talliers and auctioneers) or users. These fundamental similarities naturally beg the question:

> *What are the (black-box) relations among e-cash, e-voting and e-auction in the UC framework?*

This question is firstly of theoretical interest. Although, initiated in [IR89], a fairly complete picture of the black-box relations among most cryptographic primitives has been obtained, not much is known about the black-box relationship among *fairly complicated concrete protocols*. This direction of research may shed new light on understanding the definitions, security and complexity of these protocols.

Moreover, it is desirable to explore such relations in the modern *UC framework* introduced by Canetti [Can01]. Informally speaking, protocols secure in this framework remain secure even when executed concurrently with other arbitrary protocols running in some larger network, and can be used as subroutines of larger protocols in a modular fashion. The property is important considering that nowadays many protocols are executed concurrently with others on the Internet. We note that there are only a few results on the relationship among cryptographic primitives in the UC framework.

The question is also of practical interest. In practice, the simpler the system implementation is, the better. That is, if there is a significant black-box component that makes the implementation much simpler, there is no reason not to use it. Building a system from scratch, not employing available software in a black box manner, but using instead "smaller" black boxes (i.e., lower primitives such as one-way functions) potentially entails a lot of design and thus creates further sources for bugs in protocols or the need to embed the new protocol in a secure trusted systems component, which in turn lead to high costs (involved in resolving these issues). For example, if it is known that protocol-Y can be constructed from protocol-X, an optimized secure implementation (suitable for the setting) of protocol-X may lead to fairly simple deployment of a secure protocol-Y.

1.2 Our Results

Motivated by the above theoretical and practical considerations, we explore the relations among (off-line) e-cash, e-voting, and e-auction in the UC framework.

Concretely, we explore whether a secure protocol for e-voting (and for e-auction) can be constructed using a secure protocol for e-cash. This type of an investigation involves two tasks: defining security and achieving constructions in the UC framework.

Definition of Security. We first present ideal functionalities capturing the security of e-cash, e-voting, and e-auction; we concentrate in this work only on basic models with the most important security properties (generalizations to include further properties require extended functionalities and are left for future investigations). Intuitively, the e-cash functionality provides two important features of protection against double-spending and anonymity of honest spenders (we do not deal with extensions such as "fair cash" and "divisible cash"). The e-voting functionality provides protection against double-voting and unlinkability between voters and votes, i.e., correctness of voting and voter privacy. Again, it doesn't provide advanced properties like incoercibility [BT94,JCJ05].[4] The e-auction functionality provides secrecy of the bidding information until the end of the bidding procedure (once again, ignoring various added more advanced concerns like totally untrusted auctioneer, etc.).

E-Voting from E-Cash. We show a construction of a protocol that UC realizes the e-voting functionality in the e-cash hybrid model, under a certain restriction on the corruption pattern of the adversary. Due to the UC composition theorem [Can01], this implies that there exists a protocol π_{vote} UC-realizing the e-voting functionality with black-box access to π_{cash} UC-realizing the e-cash functionality.

We first notice similar security features between e-cash and e-voting. That is, if a voter casts a ballot more than once, his vote is rejected and possibly the identity of the double-voter is compromised (similarly to protection against double-spending in e-cash); on the other hand, voters and votes should be unlinkable (similarly to unlinkability between spenders and coins in e-cash). By utilizing these similarities (which are certainly known in the folklore) and by exploring more carefully the relationships and the needs of each problem, we were able to construct an e-voting protocol in the e-cash hybrid.

E-Auction from E-Cash. We also give a construction for e-auction (with a bound on the maximum bidding amount) in the e-cash hybrid. In the construction, there are two authorized agents, and it is assumed that at most one of them is semi-honestly corrupted. In the bidding stage, each bidder spends coins with the two authorities so that the bidding amount may be equal to the number of coins doubly-spent. Note that secrecy of the bidding amount is guaranteed since neither authority alone can determine the number of doubly-spent coins. Then, after the bidding stage ends, both authorities deposit their coins and count the number of doubly-spent coins for each bidder.

[4] In an e-voting scheme with incoercibility, it is infeasible for the adversary to determine if a coerced voter complies with the demands. We leave as an interesting open problem achieving incoercibile e-voting from e-cash.

1.3 Related Work

There have been a few results that provide an ideal functionality for e-cash [Tro05,Lin09] or e-voting [Gro04,dMPQ07]. The e-cash functionality in [Tro05] is not general enough in that the functionality contains a hash function and a tree structure inside. The e-cash functionality in [Lin09] does not deal with anonymous coins or detection of double-spending. The e-voting functionality in [Gro04] is different from ours in that it allows the adversary to prevent a voter from casting a vote while our functionality does not. The e-voting functionality in [dMPQ07] is parameterized with a post-processing function on the gathered ballots and can consider more general types of voting, e.g., outputting three most-favored candidates.

Maji, Prabhakaran, and Rosulek considered relations between cryptographic primitives in the UC framework [PR08,MPR09,MPR10a,MPR10b]. However, they have a different focus, and they rather retained more of a complexity theoretic perspectives (general feasibility) and explored which ideal functionality is complete.

1.4 Organization

In Section 2 we define ideal functionalities for e-cash, e-voting, and e-auction. Constructions of e-voting and e-auction in the e-cash hybrid are described in Section 3 and Section 4 respectively. We conclude in Section 5.

2 Ideal Functionalities

We present below the ideal functionalities for e-cash, e-voting and e-auction. We note that, although not explicitly stated in the description of the functionalities, the ideal adversary initially corrupts a subset of parties by sending a corrupt message for each party in the subset to the ideal functionality. Thus, the ideal functionality is always aware of the set of corrupted parties.

2.1 Ideal Functionality for E-Cash

We start with defining the ideal functionality $\mathcal{F}_{cash}$ for e-cash in Fig. 1. In the functionality, a user may open an account with his identity or a pseudonym *Nym* under the permission of the bank. Each coin is associated with a randomly generated serial number; when withdrawing w coins, a user is given w serial numbers, with which he can spend coins on other parties.

The functionality achieves *anonymity of honest spenders* in executing the command spend by directly notifying the serial number, but with no information about the corresponding spender, to the merchant. Note that the functionality stipulates that neither the bank nor the *merchant* should have any knowledge of who the spender is. We believe this modeling is reasonable; if merchants know the information about the spenders they may be able to sell it to the banks

Functionality $\mathcal{F}_{cash}$

– Upon receiving (setup, sid) from the prospective bank B:
  If there is already a party registered as the bank, ignore this message. Otherwise, record B as the bank. If there is no registered bank for this session sid, all the messages below are ignored.

– Upon receiving ($\mathsf{open\_account}$, sid, Nym, k) from U_i:
  If there is already an account for U_i, ignore the request. Otherwise, send ($\mathsf{open\_account}$, sid, Nym, k, $type$) to the bank, where $type$ is $\mathsf{identity}$ if $U_i = Nym$, or $\mathsf{anonymous}$ otherwise. Let ($\mathsf{opened\_account}$, sid, Nym, rep) be the reply from the bank. If $rep \neq \perp$, initialize an account for U_i tagged with Nym with initial balance k. Send ($\mathsf{opened\_account}$, sid, rep) to U_i.

– Upon receiving ($\mathsf{withdraw}$, sid, w) from U_i:
1. If there is no account for U_i or if the balance of account U_i is less than w, send ($\mathsf{withdrawn}$, sid, $\perp$) to U_i and terminate.
2. Decrease the balance of U_i by w, choose w random numbers ($serial_1, \ldots, serial_w$), and record the tuples ($withdrawn, U_i, serial_1, \ldots, serial_w$). Then, send ($\mathsf{withdrawn}$, sid, $serial_1, \ldots, serial_w$) to U_i and ($\mathsf{withdrawn}$, sid, Nym, w) to the bank B, where Nym is the tag for U_i.

– Upon receiving (spend, sid, Nym, $serial$) from U_i:
1. If there is no such record as ($withdrawn, U_i, serial$), send (spent, sid, $\perp$) to U_i and terminate.
2. Let U_j be the user whose account is tagged with Nym — if there no such user, send (spent, sid, $\perp$) to U_i and terminate. Record the tuple ($spent, U_i, U_j, serial$), send (spent, sid, Nym) to U_i, and send (spent, sid, $serial$) to U_j.

– Upon receiving ($\mathsf{deposit}$, sid, $serial$) from U_j:
1. If there is no record of a form ($spent, *, U_j, serial$) or if there is already a record ($deposited, *, U_j, serial$), then send ($\mathsf{deposited}$, sid, $\perp$) to U_j and terminate.
2. Let Nym be the tag for U_j. If there is already a record ($deposited$, U_i, U_k, $serial$) for some U_i and $U_k \neq U_j$, then record ($doubly\text{-}spent$, $serial$, U_j), send ($\mathsf{deposited}$, sid, $doubly\text{-}spent$) to U_j, send ($\mathsf{deposited}$, sid, Nym, $serial$, $doubly\text{-}spent$) to the bank, and terminate.
3. Record ($deposited, U_i, U_j, serial$) where U_i is the spender of the coin (i.e., there is a record ($spent, U_i, U_j, serial$)). Increment the balance of U_j's account. Then send ($\mathsf{deposited}$, sid, $serial$) to U_j and send ($\mathsf{deposited}$, sid, Nym, $serial$) to the bank.

– Upon receiving ($\mathsf{double\_spender}$, sid, $serial$) from the bank B:
  If there is no such record as ($doubly\text{-}spent, serial, *$), send ($\mathsf{double\_spender}$, sid, $\perp$) to B. Otherwise, find the record ($deposited$, $U_i, U_j, serial$) for some U_i and U_j, and send ($\mathsf{double\_spender}$, sid, Nym_i, Nym_j) to B, where Nym_i and Nym_j are the tags for U_i and U_j respectively.

– Upon receiving ($\mathsf{double\_spenders}$, sid) from the bank B:
  Perform as above for $serial$ such that there is a record ($doubly\text{-}spent$, $serial$, $*$). Let S be the list of the tuples ($serial$, Nym_i, Nym_j), where Nym_i is the tag for a double spender, Nym_i is the tag on which the coin $serial$ is deposited. Send ($\mathsf{double\_spender}$, sid, S) to B.

– Upon receiving ($\mathsf{balance}$, sid) from U_i:
  Let b be the balance of U_i. Send ($\mathsf{balance}$, sid, b) to U_i.

– Upon receiving ($\mathsf{balance}$, sid, Nym) from the bank B:
  Let b be the balance of the account with a tag Nym. Send ($\mathsf{balance}$, sid, b) to B.

Fig. 1: Ideal Functionality for E-Cash

and other marketing organizations, which defeats the purpose Chaum [Cha83] originally tries to achieve. Indeed, this goes in line with a standard method to achieve a secure e-cash system, that is, executing an e-cash scheme through an anonymous channel [Cha81,SGR97,RR98,SL00,DMS04].

The functionality also provides *detecting the double-spender* of a coin in an off-line manner — detection occurs when the coin is deposited. Note that the power of the bank is restricted in that it only approves the account-opening requests, observes withdrawals and deposits, and detects double spenders. Further recall that we do not model more advanced properties of various e-cash schemes beyond the simple "basic coin" model.

2.2 Ideal Functionality For Basic E-Voting

Next, we define the ideal functionality $\mathcal{F}_{vote}$ for e-voting. The functionality assumes an authority that manages the voting procedure. Each party registers for voting, and then casts a vote for his favorite candidate.

At the tallying stage, the functionality allows corrupted candidates, not knowing the voting information of others, to decrease the number of votes they have received. Since it only allows them to give up the election, the functionality regards this case as legitimate. Note also that the voting information is kept secret even to the authority until the voting stage ends. As mentioned above, note that the functionality does not consider further advanced properties desired in various election scenarios, such as incoercibility [BT94,JCJ05].

2.3 Ideal Functionality for Basic E-Auction

Finally, we formulate the ideal functionality $\mathcal{F}_{auc}$ for e-auction in Fig. 3. The functionality assumes an authority that manages the auction process. Each party registers for auction, and then casts a bid. The authority does not know the bidding information until the bidding stage ends.

In our formulation, however, the authority will eventually see all the bidding information. As mentioned above, this is the basic case we deal with in this work. Obviously, more private auctions are suitable in many scenarios, yet considering that in many practical scenarios the authority ultimately sees the bids (e.g., Google AD Exchange), we believe our modeling is still a meaningful starting point.

3 E-Voting from E-Cash

We present an e-voting protocol that UC-realizes $\mathcal{F}_{vote}$ in the $\mathcal{F}_{cash}$ hybrid, with some restrictions on the corruption pattern of an adversary. In the protocol, we employ the similar security features between e-cash and e-voting. That is, if a

Functionality $\mathcal{F}_{vote}$

Let A be the voting authority and $C_1, \ldots, C_k$ be the candidates.

– Upon receiving $(\mathsf{register}, sid)$ from V_i:

> If there is a record $(registered, V_i)$, ignore this message. Otherwise, record the tuple $(registered, V_i)$ and broadcast $(\mathsf{registered}, sid, V_i)$.

– Upon receiving (vote, sid, v) from V_i:

> If there is no such record as $(registered, V_i)$ or if $v \notin [k]$, then ignore this message. Otherwise, record the tuple $(voted, V_i, v)$.

– Upon receiving (tally, sid) from the authority A:

1. Compute the tally result $R = (r_1, \ldots, r_k)$, where r_i is the tally for candidate C_i. In computing the tally results, ignore all tuples of the form $(voted, V_j, *)$ such that V_j appears more than once.
2. For $1 \leq i \leq k$, if candidate C_i is corrupted, send message $(\mathsf{tally}, sid, r_i)$ to candidate C_i.
3. Upon receiving a message of the form $(\mathsf{tally}, sid, r'_i)$ from each corrupted candidate C_i, compute $R_{final} = R - (r''_1, \ldots, r''_k)$, where $r''_i = r'_i$ if $0 \leq r'_i \leq r_i$, or $r''_i = 0$ otherwise, and broadcast $(\mathsf{tally}, sid, R_{final})$.

Fig. 2: Ideal Functionality for E-Voting

voter casts a ballot more than once, his vote is rejected and possibly the identity of the double-voter is compromised (similarly to protection against double-spending in e-cash); on the other hand, voters and votes should be unlinkable (similarly to unlinkability between spenders and coins in e-cash).

We emphasize that the construction does not use any cryptographic tools or assumptions beyond $\mathcal{F}_{cash}$. Therefore, given a secure e-cash scheme, we can construct a secure e-voting scheme against an adversary with the specified corruption pattern.

Toy Construction. We start with a toy construction illustrating the basic idea of how to use an e-cash scheme, although with weak security.

> Some of the participants are designated as candidates. Each voter withdraws one coin from his bank account and spends his coin on the candidate of his choice. At the end of the election, each candidate deposits the coins that he receives, and broadcasts his balance as the result of the election.

In the above scheme, due to the anonymity of honest spenders of e-cash, the voting authority does not know the tallying information during the voting stage. However, the scheme is only secure against an adversary that corrupts voters maliciously and the authority semi-honestly but does not corrupt candidates. In particular, each candidate knows the exact number of votes in favor of himself.

Functionality $\mathcal{F}_{auc}$

Let A be the auction authority.

- Upon receiving $(\mathsf{register}, sid)$ from party P_i:

  If there is a record $(registered, P_i)$, ignore this message. Otherwise, record the tuple $(registered, P_i)$ and broadcast $(\mathsf{registered}, sid, P_i)$.

- Upon receiving $(\mathsf{bid}, sid, P_i, v)$ from party P_i:

  If there is no such record as $(registered, P_i)$ or if there is a record of a form $(bid, P_i, *)$, then ignore this message. Otherwise, record the tuple (bid, P_i, v).

- Upon receiving $(\mathsf{result}, sid, A)$ from the auctioneer A:

  Send all the bidding information to A, that is, $(\mathsf{result}, sid, \{(P_i, v_i)\}_{i=1}^{n})$, where n is the number of registered bidders and v_i is the bidding amount of P_i. Also, broadcast $(\mathsf{result}, sid, \mathcal{P})$, where $\mathcal{P}$ is the set of highest bidders.

Fig. 3: Ideal Functionality for E-Auction

3.1 Construction

Our final construction uses the following ideas to remove the trust trust that is placed in each of the candidates in the toy construction.

Two Authorities: The construction has two authorities: A registration authority B_1 and a tallying authority B_2.

Use of Detecting Double Spenders in E-Cash: The construction actively uses the feature of detection of double spenders in the e-cash scheme. In particular, let k be the number of candidates. Suppose a voter wants to cast a vote to the j-th candidate. Then, he withdraws k coins from the bank (i.e., the tallying authority B_2), spends each of the k coins to each candidate, and then *spends the j-th coin to the registration authority.* Each candidate deposits the coins that it received.
At the tallying stage, the registration authority deposits the coins that it has. Then the doubly spent coins are used to compute the tally result.

Use of Anonymous Spending in E-Cash: One security concern in the above description is that the identity of the voter is revealed, since the feature of detecting double spenders is used legitimately. To avoid this, each voter uses a pseudonym in spending coins. In particular, the registration authority plays a role of another bank, and the serial number of the coin with respect to the registration authority becomes a pseudonym of a voter. Since the serial number is generated at random, the pseudonym reveals no information of its owner's identity. The coins with respect to the tallying authority can now be safely used as explained above.

Protocol π_{vote}

There are two distinguished parties — registration authority B_1 and tallying authority B_2. Both play the role of a bank in the e-cash scheme. Let $C_1, \ldots, C_k$ be the candidates.

- B_1 and B_2 send $(\textsf{setup}, sid_1)$ and $(\textsf{setup}, sid_2)$ to $\mathcal{F}_{cash}$ respectively. Each candidate C_i opens an account with balance 0 with respect to B_2.
- When V_i receives a message $(\textsf{register}, sid)$ from the environment $\mathcal{Z}$:
  1. $V_i$sends $(\textsf{open\_account}, sid_1, V_i, 1)$ to $\mathcal{F}_{cash}$. Then B_1 in turn approve the $\textsf{open\_account}$ request via $\mathcal{F}_{cash}$, if V_i is a fresh, legitimate voter. Upon receiving $(\textsf{opened\_account}, sid_1, rep)$ from $\mathcal{F}_{cash}$, if $rep = \perp$, V_i terminates.
  2. V_i sends $(\textsf{withdraw}, sid_1, 1)$ to $\mathcal{F}_{cash}$. Let $(\textsf{withdrawn}, sid_1, s)$ be the reply from $\mathcal{F}_{cash}$. If $s = \perp$, V_i terminates; otherwise V_i sets $N_i = s$.
  3. V_i sends $(\textsf{spend}, sid_1, B_2, N_i)$ to $\mathcal{F}_{cash}$. If the reply from $\mathcal{F}_{cash}$ is $(\textsf{spent}, sid_1, \perp)$, V_i terminates.
  4. V_i sends $(\textsf{open\_account}, sid_2, N_i, k)$ to $\mathcal{F}_{cash}$. Then, B_2, upon receiving the $(\textsf{spent}, sid_1, N_i)$ and $(\textsf{opened\_account}, sid_2, N_i, k, \textsf{anonymous})$ from $\mathcal{F}_{cash}$, will send $(\textsf{deposit}, sid_1, B_2, N_i)$ to $\mathcal{F}_{cash}$; if the response from $\mathcal{F}_{cash}$ is $(\textsf{deposited}, sid_1, N_i)$, B_2 will send $(\textsf{open\_account}, sid_2, N_i, ok)$ to $\mathcal{F}_{cash}$; otherwise B_2 will send $(\textsf{open\_account}, sid_2, N_i, \perp)$.
- When V_i receives a message $(\textsf{vote}, sid, v)$ from the environment $\mathcal{Z}$:
  1. V_i sends $(\textsf{withdraw}, sid_2, k)$ to $\mathcal{F}_{cash}$. If $\mathcal{F}_{cash}$ responds with a message ($\textsf{withdrawn}$, sid_2, $serial_1$, …, $serial_k$), then V_i records the values $serial_1$, …, $serial_k$; otherwise, V_i terminates.
  2. For $1 \leq \ell \leq k$, V_i sends $(\textsf{spend}, sid_2, C_\ell, serial_\ell)$ to $\mathcal{F}_{cash}$. If $\mathcal{F}_{cash}$ responds with a message $(\textsf{spent}, sid_2, \perp)$ for any of the requests, V_i terminates.
  3. V_i sends $(\textsf{spend}, sid_2, B_1, serial_v)$ to $\mathcal{F}_{cash}$. If $\mathcal{F}_{cash}$ responds with a message $(\textsf{spent}, sid_2, \perp)$, V_i terminates.
  4. Now, each candidate C_ℓ, upon receiving a message $(\textsf{spent}, sid_2, s)$ from $\mathcal{F}_{cash}$, sends a message $(\textsf{deposit}, sid_2, s)$ to $\mathcal{F}_{cash}$.
- When B_1 receives a message $(\textsf{tally}, sid)$ from $\mathcal{Z}$:
  1. B_1 sends a message ($\textsf{deposit}$, sid_2, s) to $\mathcal{F}_{cash}$ for each s of the coins that it has received; while B_1 is doing so, B_2 records the serial number s if B_2 receives $(\textsf{deposited}, sid_2, B_1, s, \textit{doubly-spent})$ from $\mathcal{F}_{cash}$. B_1 sends $(\textsf{tally}, sid)$ to B_2.
  2. For each recorded serial number s, B_2 sends $(\textsf{double\_spender}, sid_2, s)$ to $\mathcal{F}_{cash}$ and retrieves the corresponding pseudonym (i.e., the double spender) and the corresponding candidate. Using this list, B_2 computes the tally $R = (r_1, \ldots, r_k)$ and broadcasts $(\textsf{tally}, sid, R)$. If a pseudonym appears more than once in the list, B_2 ignores all the votes given by the pseudonym.
  3. Each party outputs R.

Fig. 4: The Protocol for E-Voting in the $\mathcal{F}_{cash}$-Hybrid

The description of the overall protocol π_{vote} is given in Figure 4.

3.2 Security

The authorities, once corrupted, are assumed to behave in a semi-honest manner. Also, we consider the case where at most one of the authorities is corrupted by the adversary. A more serious restriction is that the adversary is not allowed to corrupt the registration authority and candidates at the same time. If both the registration authority and a candidate have received a coin with the same serial number, it means that someone voted for the candidate. Therefore, such corruptions reveal to the adversary the number of votes casted for the corrupted voters before the tallying stage. We believe this restriction on the authorities is reasonable.

Theorem 1. *The protocol π_{vote} UC-realizes $\mathcal{F}_{vote}$ functionality against an adversary that corrupts voters and candidates maliciously and the authorities semi-honestly, with the restriction that it is not allowed to corrupt the registration authority and some candidates at the same time.*

Proof. We show that for every adversary $\mathcal{A}$, there exists a PPT $\mathcal{S}$ such that for every non-uniform PPT $\mathcal{Z}$ it holds that

$$\text{EXEC}^{\mathcal{F}_{cash}}_{\pi_{vote},\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{vote},\mathcal{S},\mathcal{Z}}.$$

Fix $\mathcal{A}$. Wlog, we assume B_2 is semi-honestly corrupted; the proof is only easier when B_2 is not corrupted. We consider two cases according to whether B_1 is corrupted or not.

We will refer to the communication of $\mathcal{S}$ with $\mathcal{Z}$ and $\mathcal{F}_{vote}$ as external communication, and that with $\mathcal{A}$ as internal communication. For clarity, the message exchanges between the real adversary $\mathcal{A}$ (on behalf of a corrupted party P) and the simulator $\mathcal{S}$ (simulating ideal functionality $\mathcal{F}_{cash}$) are represented as exchanges between P and $\mathcal{S}$.

Roughly speaking, the simulator $\mathcal{S}$ simulates the functionality $\mathcal{F}_{cash}$ internally and tries to extract votes from corrupted voters.

Throughout the running of $\mathcal{S}$, it $\mathcal{S}$ forwards all the messages between $\mathcal{A}$ and $\mathcal{Z}$. Below, we describe how $\mathcal{S}$ handles other messages.

Case 1: B_1 is not corrupted. Let's first consider the case where B_1 is not corrupted. Then, the adversary may corrupt some candidates as well.

Handling register. $\mathcal{S}$ handles register messages as follows:

- $\mathcal{S}$, as $\mathcal{F}_{cash}$ and B_1 in the internal communication, handles the following messages exactly as $\mathcal{F}_{cash}$ and B_1 would do:
  - (setup, sid_2) from B_2.
  - (open_account, sid_1, V_i, bal) from a corrupted V_i.
  - (withdraw, sid_1, w) from a corrupted V_i.
  - (spend, $sid_1, B_2, serial$) from a corrupted V_i.
  - (deposit, $sid_1, serial$) from B_2.

- When $\mathcal{S}$ as $\mathcal{F}_{cash}$ receives $(\mathsf{open\_account}, sid_2, serial, k)$ from a corrupted V_i:
  $\mathcal{S}$ does exactly what $\mathcal{F}_{cash}$ would do. In addition, when B_2 approves the request with $(\mathsf{opened\_account}, sid_2, ok)$, then $\mathcal{S}$, as the corrupted V_i in the external communication, sends $(\mathsf{register}, sid)$ to $\mathcal{F}_{vote}$.
- When $\mathcal{S}$ receives $(\mathsf{registered}, sid, V_j)$ externally from $\mathcal{F}_{vote}$ for an uncorrupted V_j:
  $\mathcal{S}$ does exactly what V_j will do in the internal communication. In particular, $\mathcal{S}$ handles virtual register, open_account, withdraw, and spend messages for V_j, as specified in the protocol.

Handling vote. $\mathcal{S}$ handles vote messages as follows:

- $\mathcal{S}$ as $\mathcal{F}_{cash}$ handles the following messages exactly as $\mathcal{F}_{cash}$ would do:
  - $(\mathsf{withdraw}, sid_2, k)$ from a corrupted V_i.
  - $(\mathsf{spend}, sid_2, B_1, s)$ from V_i.
  - $(\mathsf{spend}, sid_2, C_j, s)$ from V_i.
- When $\mathcal{S}$ as $\mathcal{F}_{cash}$ receives $(\mathsf{deposit}, sid_2, s)$ from a corrupted candidate C_j:
  $\mathcal{S}$ behaves exactly $\mathcal{F}_{cash}$ would. In addition, if the coin s has been successfully deposited and if the coin s has also been spent on B_1, find the spender V_i of the coins s using the recorded data, and $\mathcal{S}$ as the corrupted V_i (in the external communication), externally sends a message (vote, sid, v) to $\mathcal{F}_{vote}$.
- For each uncorrupted voter V_i, $\mathcal{S}$ simulates its behavior as specified in the protocol. The only exception is that V_i delays spending a coin on B_1 until the tally result comes out. This is because $\mathcal{S}$ cannot know which vote to cast on behalf of the honest party until it receives the final tally from the ideal voting functionality.
- For each uncorrupted candidate C_i, $\mathcal{S}$ simulates its behavior as specified in the protocol. In addition, if C_i has successfully deposited a coin s and if the coin s has also been spent on B_1, find the spender V_i of the coins s using the recorded data, and $\mathcal{S}$ as the corrupted V_i (in the external communication), externally sends a message (vote, sid, v) to $\mathcal{F}_{vote}$.

Handling tally. $\mathcal{S}$ handles tally messages as follows:

- When $\mathcal{S}$, as a corrupted candidate C_i (in the external communication), externally receives $(\mathsf{tally}, sid, r_i)$ from $\mathcal{F}_{vote}$:
  1. Let m_i be the number of votes given by the corrupted voters to C_i, in the internal communication. This can be computed from the recorded data.
  2. Let $h_i = r_i - m_i$; that is, h_i is the number of votes from uncorrupted voters. $\mathcal{S}$ arbitrary generate a set H_i (disjoint with H_j for other candidate C_j) of uncorrupted voters of size h_i. On behalf of each uncorrupted voter V in H_i, $\mathcal{S}$ now handles a virtual message $(\mathsf{spend}, sid_2, B_1, s')$ from V, where s' is the coin that V has spent on C_i. Note that this simulation is good since $\mathcal{F}_{cash}$ guarantees that identities of spenders and serial numbers of coins are completely disassociated.

3. Let h'_i be the number of coins belonging to H_i that C_i didn't deposit. $\mathcal{S}$, as the corrupted C_i (in the external communication), externally sends $(\mathsf{tally}, sid, h'_i)$ to $\mathcal{F}_{vote}$.

– Once $\mathcal{S}$ handles the tally messages from $\mathcal{F}_{vote}$ for all the corrupted candidates in the external communication, $\mathcal{S}$ as B_1 internally sends (tally, sid) to B_2.
– When $\mathcal{S}$ as $\mathcal{F}_{cash}$ receives a message $(\mathsf{double\_spender}, sid_2, s)$ from B_2:
  $\mathcal{S}$ does exactly what $\mathcal{F}_{cash}$ would do.

Case 2: When B_1 is corrupted. In this case, due the restriction on the corruption pattern of the adversary, the candidates are not to be corrupted.

Handling register. $\mathcal{S}$ handles register messages as follows:

– $\mathcal{S}$ as $\mathcal{F}_{cash}$ handles the following messages as $\mathcal{F}_{cash}$ would do:
  - setup messages from B_1 or B_2.
  - $(\mathsf{open\_account}, sid_1, V_i, bal)$ from a corrupted V_i.
  - $(\mathsf{withdraw}, sid_1, w)$ from a corrupted V_i.
  - $(\mathsf{spend}, sid_1, B_2, serial)$ from a corrupted V_i.
  - $(\mathsf{deposit}, sid_1, serial)$ from B_2.
– When $\mathcal{S}$ as $\mathcal{F}_{cash}$ receives $(\mathsf{open\_account}, sid_2, serial, k)$ from a corrupted V_i:
  $\mathcal{S}$ does exactly what $\mathcal{F}_{cash}$ would do. In addition, when B_2 approves the request with $(\mathsf{opened\_account}, sid_2, ok)$, $\mathcal{S}$ as the corrupted V_i (in the external communication) externally sends $(\mathsf{register}, sid)$ to $\mathcal{F}_{vote}$.
– When $\mathcal{S}$ receives $(\mathsf{registered}, sid, V_j)$ from $\mathcal{F}_{vote}$:
  $\mathcal{S}$ internally simulates what the uncorrupted V_j would do. In particular, $\mathcal{S}$ handles virtual register, open_account, withdraw, and spend messages for V_j, as specified in the protocol.

Handling vote. $\mathcal{S}$ handles vote messages as follows:

– $\mathcal{S}$ as $\mathcal{F}_{cash}$ handles the following messages as $\mathcal{F}_{cash}$ would do:
  - $(\mathsf{withdraw}, sid_2, k)$ from a corrupted V_i.
  - $(\mathsf{spend}, sid_2, B_1, s)$ from a corrupted V_i.
  - $(\mathsf{spend}, sid_2, C_j, s)$ from a corrupted V_i:
– For each uncorrupted voter V_i, $\mathcal{S}$ simulates its behavior as specified in the protocol, except the following:
  Let $s_1, \ldots, s_k$ be the coins that V_i has. V_i spends s_1 on B_1 and delays spending coins on candidates until the tally result comes out.
– For each uncorrupted candidate C_i, $\mathcal{S}$ simulates its behavior as specified in the protocol.
– If C_i, whether or not it is corrupted, has successfully deposited a coin s and if the coin s has also been spent on B_1, find the spender V_i of the coins s using the recorded data, and $\mathcal{S}$ as the corrupted V_i (in the external communication), externally sends a message (vote, sid, v) to $\mathcal{F}_{vote}$.

Handling tally. $\mathcal{S}$ handles tally messages as follows:

– When $\mathcal{S}$, as $\mathcal{F}_{cash}$, starts to get $\mathsf{deposit}$ requests from B_1, $\mathcal{S}$, as the corrupted registration authority (in the external communication) sends (tally, sid) to $\mathcal{F}_{vote}$. The $\mathsf{deposit}$ requests are handled as $\mathcal{F}_{cash}$ would do.
– Upon receiving $(\mathsf{tally}, sid, R_{\mathit{final}})$ externally from $\mathcal{F}_{vote}$:
   1. Let $R_{\mathit{final}} = (r_1, \ldots, r_k)$. For each candidate C_i, let m_i be the number of votes given by corrupted voters to C_i in the internal communication. This can be computed from the recorded data. Let $h_i = r_i - m_i$; that is, h_i is the number of votes from uncorrupted voters. $\mathcal{S}$ arbitrary generate a set H_i (disjoint with H_j for other candidate C_j) of uncorrupted voters of size h_i. Now, each uncorrupted voter V in H_i spends coins on candidates. Let $s_1, \ldots, s_k$ be the coins that V has. V spends $(s_1, s_2, s_3, \ldots, s_k)$ on $(C_i, C_2, C_3, \ldots, C_{i-1}, C_1, C_{i+1}, \ldots C_k)$.
– When $\mathcal{S}$ as $\mathcal{F}_{cash}$ receives a message $(\mathsf{double\_spender}, sid_2, s)$ from B_2:
   $\mathcal{S}$ does exactly what $\mathcal{F}_{cash}$ would do.

4 E-Auction from E-Cash

We construct a protocol π_{auc} for e-auction with a bound on the maximum bidding amount in the $\mathcal{F}_{cash}$-hybrid that UC-realizes the $\mathcal{F}_{auc}$ functionality in Fig. 5.

Overview of the Protocol. In the protocol, there are two authorized agents A_1 and A_2, which will play the role of a bank in the e-cash scheme. We note that A_1 and A_2 are assumed to be semi-honest and are trusted not to collude (i.e., we allow at most one of them to be corrupted).

Let θ be the maximum bidding value. Each player withdraws 2θ coins from the first authority A_1 and spends θ coins on A_1 and A_2 respectively. The idea is that each party spends coin, using the feature of detecting the double spenders in $\mathcal{F}_{cash}$, so that the bidding amount may be equal to the number of coins doubly-spent. For example, party P with bidding amount k will spend the k coins (out of θ) on both A_1 and A_2. Then, after the bidding stage ends, A_1 and A_2 will deposit their coins and count the number of doubly-spent coins for each bidder.

Theorem 2. *The protocol π_{auc} UC-realizes $\mathcal{F}_{auc}$ functionality as long as at most one of A_1 and A_2 is semi-honestly corrupted.*

Proof. We show that for every adversary $\mathcal{A}$, there exists a PPT $\mathcal{S}$ such that for every non-uniform PPT $\mathcal{Z}$ it holds that

$$\mathrm{EXEC}^{\mathcal{F}_{cash}}_{\pi_{auc},\mathcal{A},\mathcal{Z}} \approx \mathrm{IDEAL}_{\mathcal{F}_{auc},\mathcal{S},\mathcal{Z}}.$$

Fix $\mathcal{A}$. At the outset of the protocol, there are some bidders that are corrupted. We consider the case where A_1 is semi-honestly corrupted; the case where A_1 is not corrupted is only easier. We will refer to the communication of $\mathcal{S}$ with $\mathcal{Z}$ and $\mathcal{F}_{auc}$ as external communication, and that with $\mathcal{A}$ as internal communication.

Protocol π_{auc}

Let θ be the maximum bidding value and A_1 and A_2 be auction authorities.

- A_1 sends (setup, sid) to $\mathcal{F}_{cash}$. A_1 and A_2 open their accounts respectively.
- When P_i receives a message (register, sid) from the environment $\mathcal{Z}$:
  1. P_i sends (open_account, $sid, P_i, 2\theta$) to $\mathcal{F}_{cash}$. Then A_1 in turn will receive the open_account request from $\mathcal{F}_{cash}$ and, if P_i is legitimate, approve the request.
  2. Upon receiving (opened_account, sid, rep) from $\mathcal{F}_{cash}$, if $rep = \perp$, P_i terminates.
  3. P_i sends (withdraw, $sid_1, P_i, A_1, 2\theta$) to $\mathcal{F}_{cash}$. If $\mathcal{F}_{cash}$ responds with a message (withdrawn, sid_1, $serial_1$, …, $serial_{2\theta}$), then P_i records the values $serial_1, \ldots, serial_{2\theta}$.
- When P_i receives a message (bid, sid, v) from the environment $\mathcal{Z}$:
  1. If P_i does not have a recorded $serial_1, \ldots, serial_{2\theta}$, it terminates.
  2. For $1 \leq j \leq \theta$, P_i sends a message (spend, $sid, P_i, A_1, serial_j$) to $\mathcal{F}_{cash}$. If $\mathcal{F}_{cash}$ responds with a message (spent, $sid, \perp$) to any of the requests, P_i terminates.
  3. For $\theta - v + 1 \leq j \leq 2\theta - v$, P_i sends a message (spend, $sid, P_i, A_2, serial_j$) to $\mathcal{F}_{cash}$. If $\mathcal{F}_{cash}$ responds with a message (spent, $sid, \perp$) to any of the requests, P_i terminates.
- When A_1 receives a message (result, sid) from $\mathcal{Z}$:
  1. A_1 tells A_2 to deposit its coins. After A_2 has finished the deposit procedure, A_1 deposits all the coins. For each serial number $serial_j$ that has been doubly-spent, A_1 sends a message (double_spender, $sid, serial_j$) to $\mathcal{F}_{cash}$ and receives back the identity of the spender of coin $serial_j$. Now, using this information, A_1 determines the bidding amount of each participant P_k. A_1 broadcasts the message (result, $sid, \mathcal{P}$), where $\mathcal{P}$ is the set of parties whose bidding amount is the largest.

Fig. 5: Protocol for E-Auction in the $\mathcal{F}_{cash}$-hybrid

Roughly speaking, the simulator $\mathcal{S}$ simulates the functionality $\mathcal{F}_{cash}$ internally and tries to extract bids from corrupted bidders.

For clarity, message exchanges between the real adversary $\mathcal{A}$ (on bahalf of a corrupted party P) and the simulator $\mathcal{S}$ (simulating ideal functionality $\mathcal{F}_{cash}$) are represented as exchanges between P and $\mathcal{S}$.

Throughout the running of $\mathcal{S}$, it $\mathcal{S}$ forwards all the messages between $\mathcal{A}$ and $\mathcal{Z}$. Below, we describe how $\mathcal{S}$ handles other messages.

Handling register.

The simulator $\mathcal{S}$ handles the register messages as follows:

- When $\mathcal{S}$ as $\mathcal{F}_{cash}$ receives (open_account, $sid_1, P_i, 2\theta$) from a corrupted bidder P_i:

  $\mathcal{S}$ does exactly what $\mathcal{F}_{cash}$ would do.

- When $\mathcal{S}$ as $\mathcal{F}_{cash}$ receives $(\mathsf{withdraw}, sid_1, P_i, w)$ from a corrupted P_i:
  $\mathcal{S}$ does exactly what $\mathcal{F}_{cash}$ would do. In addition, for a successful withdrawal, $\mathcal{S}$, as P_i in the external communication, sends $(\mathsf{register}, sid)$ externally to $\mathcal{F}_{auc}$.
- When $\mathcal{S}$ receives $(\mathsf{registered}, sid, P_j)$ externally from $\mathcal{F}_{auc}$ for an uncorrupted P_j:
  $\mathcal{S}$ does exactly what P_j will do in the internal communication. In particular, $\mathcal{S}$ as P_j sends $(\mathsf{open\_account}, sid_1, P_i, 2\theta)$ to $\mathcal{F}_{cash}$ ($\mathcal{S}$ itself) and handles the virtual withdraw message (i.e., sends the message to $\mathcal{S}$ itself).

Handling bid.
The simulator $\mathcal{S}$ handles the bid messages as follows:

- When $\mathcal{S}$ as $\mathcal{F}_{cash}$ receives $(\mathsf{spend}, sid_1, P_i, A_j, serial)$ where $j \in \{1, 2\}$ from a corrupted P_i:
  $\mathcal{S}$ does exactly what $\mathcal{F}_{cash}$ would do:
- $\mathcal{S}$ handles virtual spend messages from the uncorrupted parties. That is, for each uncorrupted P_j, let $serial_1, \ldots, serial_{2\theta}$ be the serial numbers of the coins belonging to P_j. $\mathcal{S}$ as $\mathcal{F}_{cash}$ sends $\left\{(\mathsf{spent}, P_j, A_1, serial_i)\right\}_{i=1}^{\theta}$ to A_1, and it also sends $\left\{(\mathsf{spent}, P_j, A_2, serial_i)\right\}_{i=\theta+1}^{2\theta}$ to A_2.

Handling result.
The simulator $\mathcal{S}$ handles the result messages as follows:

- When $\mathcal{S}$ as A_2 receives a message from the corrupted A_1 to deposit the coins:
  1. For each corrupted bidder P_i, $\mathcal{S}$ computes the number b of doubly-spent coins by P_i, and, as P_i in the external communication, sends (bid, sid, b) to $\mathcal{F}_{auc}$ externally.
  2. $\mathcal{S}$ as the corrupted A_1 (in the external communication) sends $(\mathsf{result}, sid, A_1)$ externally to $\mathcal{F}_{auc}$ and gets the results $(\mathsf{result}, sid, \{(P_i, v_i)\}_{i=1}^{n})$. Now, $\mathcal{S}$ changes the serial numbers spent by uncorrupted parties P_j so that the number of doubly-spend coins by P_j are v_j. Then $\mathcal{S}$ handles the virtual deposit messages — i.e., sends $(\mathsf{deposit}, sid, A_2, serial)$ messages to itself — exactly as $\mathcal{F}_{cash}$ would do, using the recorded data.
- When $\mathcal{S}$ as $\mathcal{F}_{cash}$ receives deposit and double_spender messages from the corrupted A_1, $\mathcal{S}$ handles the messages exactly $\mathcal{F}_{cash}$ would do using the recorded data.

The only modification that $\mathcal{S}$ performs lies in the serial numbers of the uncorrupted parties spend on A_2. However, to the view of the adversary (in particular to A_1), this modification is invisible. Therefore, the simulation is perfect.

The case where A_1 is honest and A_2 is semi-honestly corrupted can be simulated in a similar fashion. In particular, the simulator extracts the bidding amount of corrupted bidders while simulating $\mathcal{F}_{cash}$. Then, upon receiving $(\mathsf{result}, sid, \mathcal{P})$ from $\mathcal{F}_{auc}$, $\mathcal{S}$, as A_1 in the internal communication, internally broadcasts $(\mathsf{result}, sid, \mathcal{P})$.

5 Conclusions

Our work reveals interesting relationships between some basic protocols that have been so far developed independently. Natural questions remain:

– Is it possible to eliminate the restriction on the corruption pattern of the adversary that the current constructions have or to show a separation when an arbitrary number of parties may be corrupt?
– It is also interesting to explore the remaining relationships among e-cash, e-voting, and e-auction, to consider extended functionalities (with extra properties) and explore relationships among them.
– Are there other "partial information protocols" that can be used as building blocks for other protocols or can be built on top of some known protocols?

References

[BT94] Josh Cohen Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *STOC*, pages 544–553, 1994.

[BY86] Josh Cohen Benaloh and Moti Yung. Distributing the power of a government to enhance the privacy of voters (extended abstract). In *PODC*, pages 52–62, 1986.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[CF85] Josh D. Cohen and Michael J. Fischer. A robust and verifiable cryptographically secure election scheme (extended abstract). In *FOCS*, pages 372–382, 1985.

[CFN88] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *CRYPTO*, pages 319–327, 1988.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[Cha83] David Chaum. Blind signature system. In *CRYPTO*, page 153, 1983.

[dMPQ07] Olivier de Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Simulation-based analysis of e2e voting systems. In *Frontiers of Electronic Voting*, 2007.

[DMS04] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The second-generation onion router. In *Proc. of the 13th. USENIX Security Symposium*, August 2004.

[FR96] Matthew K. Franklin and Michael K. Reiter. The design and implementation of a secure auction service. *IEEE Trans. Software Eng.*, 22(5):302–312, 1996.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[Gro04] Jens Groth. Evaluating security of voting schemes in the universal composability framework. In *ACNS*, pages 46–60, 2004.

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *STOC*, pages 44–61, 1989.

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *WPES*, pages 61–70, 2005.

[Lin09] Yehuda Lindell. Legally enforceable fairness in secure two-party communication. *Chicago J. Theor. Comput. Sci.*, 2009, 2009.

[MPR09] Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. Complexity of multi-party computation problems: The case of 2-party symmetric secure function evaluation. In *TCC*, pages 256–273, 2009.

[MPR10a] Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. Cryptographic complexity classes and computational intractability assumptions. In *ICS*, pages 266–289, 2010.

[MPR10b] Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. A zero-one law for cryptographic complexity with respect to computational uc security. In *CRYPTO*, pages 595–612, 2010.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce*, pages 129–139, 1999.

[PR08] Manoj Prabhakaran and Mike Rosulek. Cryptographic complexity of multi-party computation problems: Classifications and separations. In *CRYPTO*, pages 262–279, 2008.

[RR98] M.K. Reiter and A.D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, November 1998.

[SGR97] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous connections and onion routing. In *IEEE Symposium on Security and Privacy*, pages 44–54, Oakland, California, 1997.

[SL00] C. Shields and B. Levine. A Protocol for Anonymous Communication over the Internet. In *Proc. 7th ACM Conference on Computer and Communication Security*, November 2000.

[Tro05] Mårten Trolin. A universally composable scheme for electronic cash. In *INDOCRYPT*, pages 347–360, 2005.

[Yao86] Andrew Chi-Chih Yao. How to generate an exchange secrets. In *FOCS*, pages 162–167, 1986.