Formal Analysis of E-Cash Protocols*

Jannik Dreier[1], Ali Kassem[2] and Pascal Lafourcade[3]
[1]Institute of Information Security, Department of Computer Science, ETH, Zurich, Switzerland
[2]University Grenoble Alpes, Verimag, Grenoble, France
[3]University Clermont Auvergne, Limos, France

Keywords: E-Cash, Formal Analysis, Double Spending, Exculpability, Privacy, Applied π-Calculus, ProVerif.

Abstract: Electronic cash (e-cash) aims at achieving client privacy at payment, similar to real cash. Several security protocols have been proposed to ensure privacy in e-cash, as well as the necessary unforgery properties. In this paper, we propose a formal framework to define, analyze, and verify security properties of e-cash systems. To this end, we model e-cash systems in the applied π-calculus, and we define two client privacy properties and three properties to prevent forgery. Finally, we apply our definitions to an e-cash protocol from the literature proposed by Chaum *et al.*, which has two variants and a real implementation based on it. Using ProVerif, we demonstrate that our framework is suitable for an automated analysis of this protocol.

1 INTRODUCTION

Although current banking and electronic payment systems such as credit cards or, *e.g.*, PayPal allow clients to transfer money around the world in a fraction of a second, they do not fully ensure the clients' privacy. In such systems, no transaction can be made in a completely anonymous way, since the bank or the payment provider knows the details of the clients' transactions. By analyzing a client payments for, *e.g.*, transportations, hotels, restaurants, movies, clothes, and so on, the payment provider can typically deduce the client's whereabouts, and much information about his lifestyle.

Physical cash provides better privacy: the payments are difficult to trace as there is no central authority that monitors all transactions, in contrast to most electronic payment systems. This property is the inspiration for "untraceable" e-cash systems. The first such e-cash system preserving the client's anonymity was presented by David Chaum (Cha83): a client can withdraw a coin anonymously from his bank and spend it with a seller. The seller can then deposit the coin at the bank, who will credit his account. In this protocol coins are *non-transferable*, *i.e.*, the seller cannot spend a received coin again, but has to deposit it at the bank. If he wants to spend a coin in another shop, he has to withdraw a new coin from his account, similar to the usual payment using cheques. In contrast, there are protocols where coins are also *transferable*, *i.e.*, coins do not need to be deposited directly after each spend, but can be used again, *e.g.,* (OO89; CGT08).

To be secure, an e-cash protocol should not only ensure the client's privacy, but must also ensure that a client cannot *forge* coins which were not issued by the bank. Moreover, it must protect against *double spending* – otherwise a client could try to use the same coin multiple times. This can be achieved by using online payments, *i.e.,* a seller has to contact the bank at payment before accepting the coin, however it is an expensive solution. An alternative solution, which is usually used to support off-line payments (*i.e.,* a seller can accept the payment without contacting the bank), is revealing the client's identity if he spent a coin twice. Finally, *exculpability* ensures that an attacker cannot forge a double spend, and hence incorrectly blame an honest client for double spending.

In the literature, many e-cash protocols have been proposed (Cha83; CFN90; Dam90; DC94; Cre94; Bra94; AF96; KO01; FHY13). For example, Abe *et al.* (AF96) introduced a scheme based on partial blind signature, which allows the signer (the bank) to include certain information in the blind signature of the coin, for example the expiration date or the value of the coin. Kim *et al.* (KO01) propose an e-cash system that supports coin refund and assigns them a

*This research was conducted with the support of the "Digital trust" Chair from the University of Auvergne Foundation.

Dreier J., Kassem A. and Lafourcade P..
Formal Analysis of E-Cash Protocols.
DOI: 10.5220/0005544500650075
In *Proceedings of the 12th International Conference on Security and Cryptography* (SECRYPT-2015), pages 65-75
ISBN: 978-989-758-117-5

value, based again on partial blind signature.

At the same time, many attacks have been found against various existing e-cash protocols: for example Pfitzmann *et al.* (PW91; PSW95) break the anonymity of (Dam90; DC94; Cre94). Cheng *et al.* (CYS05) show that Brand's protocol (Bra94) allows a client to spend a coin more than once without being identified. Aboud *et al.* (AA14) show that (FHY13) cannot ensure the anonymity and unlinkability properties that were claimed.

These numerous attacks triggered some first work on formal analysis of e-cash protocols in the computational (CG08) and symbolic world (LCPD07; SK14). Canard *et al.* (CG08) provide formal definitions for various privacy and unforgeability properties in the computational world, but only with manual proofs as their framework is difficult to automate. In contrast, Luo *et al.* (LCPD07) and Thandar *et al.* (SK14) both rely on automatic tools (AVISPA[2] and ProVerif (Bla01), respectively). Yet, they only consider a fraction of the essential security properties, and for some properties Thandar *et al.* only perform a manual analysis. Moreover, much of their reasoning is targeted on their respective case studies, and cannot easily be transferred to other protocols.

Contributions: This paper fills the gaps of previous formal verification work. Inspired by other domains such as e-voting (BHM08; DKR09; DLL12), e-auctions (DLL13), and e-exams (DGK$^+$14), we propose a more general formalization for non-transferable e-cash protocols in the applied π-calculus (AF01). Our definitions are amenable to automatic verification using ProVerif (Bla01), and cover all crucial privacy and unforgery properties: *Weak Anonymity*, *Strong Anonymity*, *Unforgeability*, *Double Spending Identification*, and *Exculpability*. Finally, we validate our approach by analyzing the online protocol proposed by Chaum *et al.* (Cha83), as well as, a real implementation based on it (Sch97). We also analyze the off-line variant of this e-cash system (CFN90).

Outline: In Section 2, we model e-cash protocols in the applied pi-calculus. Then, we specify the security properties in Section 3. We validate our framework by analyzing the on-line and off-line e-cash systems by Chaum *et al.* (Cha83; CFN90), and the implementation based on the on-line protocol (Sch97) in Section 4. In Section 5, we discuss our results and outline future work.

2 MODELING E-CASH PROTOCOLS

We model e-cash protocols in the applied π-calculus, a process calculus designed for the verification of cryptographic protocols. We refer to the original paper (AF01) for a detailed description of its syntax and semantics.

In the applied π-calculus, we have a Dolev-Yao style attacker (DY83), which has a complete control to the network, except the private channels. He can eavesdrop, remove, substitute, duplicate and delay messages that the parties are sending to one another, and even insert messages of his choice on the public channels.

Parties other than the attacker can be either honest or corrupted. Honest parties follow the protocol's specification, do not reveal their secret data (*e.g.,* account numbers, keys etc.) to the attacker, and do not take malicious actions such as double spending a coin or generating fake transactions. Honest parties are modeled as processes in the applied π-calculus. These processes can exchange messages on public or private channels, create fresh random values and perform tests and cryptographic operations, which are modeled as functions on terms with respect to an equational theory describing their properties.

Corrupted parties are those that collude with the attacker by revealing their secret data to him, taking orders from him, and also making malicious actions. We model corrupted parties as in Definition 15 from (DKR09): if the process P is an honest party, then the process P^c is its corrupted version. This is a variant of P which shares with the attacker channels ch_1 and ch_2. Through ch_1, P^c sends all its inputs and freshly generated names (but not other channel names). From ch_2, P^c receives messages that can influence its behavior.

An e-cash system involves the following parties: the *client* C who has an account at the bank, the *seller* S who accepts electronic coins, and the bank B, which certifies the electronic coins. E-cash protocols typically run in three phases:

1. **Withdrawal:** the client withdraws an electronic coin from the bank, which debits the client's account.
2. **Payment:** the client spends the coin by executing a transaction with a seller.
3. **Deposit:** the seller deposits the transaction at the bank, which credits the seller's account.

In addition to these three main phases, some systems allow the clients

[2] www.avispa-project.org

(a) to return coins directly to the bank without using them in a payment, for instance in case of expiration, or to re-distribute the coins denominations, and

(b) to restore coins that have been lost, for instance due to a hard disk crash.

As these functionalities are not implemented by all protocols, our model does not require them. Moreover, we assume that the coins are neither transferable nor divisible.

We define an e-cash protocol as a tuple of processes each representing the role of a certain party.

Definition 1. (E-cash Protocol). *An* e-cash protocol *is a tuple* $(B,S,C,\tilde{n})$*, where B is the process executed by the bank, S is the process executed by the sellers, C is the process executed by the clients, and ñ is the set of the private channel names used by the protocol.*

To reason about privacy properties we use runs of the protocol, called e-cash instances.

Definition 2. (E-cash Instance). *Given an e-cash protocol, an* e-cash instance *is a closed plain process:*

$$\begin{aligned} CP = \nu\tilde{n'}.(B|S\sigma_{ids_1}|\dots|S\sigma_{ids_l}|\\ (C\sigma_{idc_1}\sigma_{c_{11}}\sigma_{ids_{11}}|\dots|C\sigma_{idc_1}\sigma_{c_{1p_1}}\sigma_{ids_{1p_1}})|\\ \vdots\\ |(C\sigma_{idc_k}\sigma_{c_{k1}}\sigma_{ids_{k1}}|\dots|C\sigma_{idc_k}\sigma_{c_{kp_k}}\sigma_{ids_{kp_k}})) \end{aligned}$$

where $\tilde{n'}$ *is the set of all restricted names which includes the set of the protocol's private channels ñ; B is the process executed by the bank;* $S\sigma_{ids_i}$ *is the process executed by the seller whose identity is specified by the substitution* σ_{ids_i}*;* $C\sigma_{idc_i}\sigma_{c_{ij}}\sigma_{ids_{ij}}$ *is the process executed by the client whose identity is specified by the substitution* σ_{idc_i}*, and which spends the coin identified by the substitution* $\sigma_{c_{ij}}$ *to pay the seller with the identity specified by the substitution* $\sigma_{ids_{ij}}$*. Note that* idc_i *can spend* p_i *coins.*

To improve the readability of our definitions, we introduce the notation of context $CP_I[\_]$ to denote the process CP with "holes" for all processes executed by the parties whose identities are included in the set I. For example, to enumerate all the sessions executed by the client idc_1 without repeating the entire e-cash instance, we can rewrite CP as $CP_{\{idc_1\}}[C\sigma_{idc_1}\sigma_{c_{11}}\sigma_{ids_{11}}|\dots|C\sigma_{idc_1}\sigma_{c_{1p_1}}\sigma_{ids_{1p_1}}]$.

Finally, we use the notation C_w to denote a client that withdraws a coin, but does not spend it in a payment: C_w is a variant of the process C that halts at the end of withdrawal phase, i.e., where the code corresponding to the payment phase is removed.

3 SECURITY PROPERTIES

We define three properties related to forgery: *Unforgeability*, *Double Spending Identification*, and *Exculpability*. Moreover, we formalize two privacy properties: *Weak Anonymity* and *Strong Anonymity*.

3.1 Forgery-related Properties

In an e-cash protocol a client must not be able to create a coin without involving the bank, resulting in a fake coin, or to double spend a valid coin he withdrew from the bank. This is ensured by *Unforgeability*, which says that the clients cannot spend more coins than they withdrew.

To define unforgeability we use the following two events:

- *withdraw(c)*: is an event emitted when the coin *c* is withdrawn. This event is placed inside the bank process just after the bank outputs the coin's certificate (*e.g.,* a signature on the coin).
- *spend(c)*: is an event emitted when the coin *c* is spent. This event is placed inside the seller process just after he receives and accepts the coin.

Events are annotations that mark important steps in the protocol execution, but do otherwise not change the behavior of processes.

Definition 3. (Unforgeability). *An e-cash protocol ensures* Unforgeability *if, for every e-cash instance CP, each occurrence of the event* spend(c) *is preceded by a distinct occurrence of the event* withdraw(c) *on every execution trace.*

If a fake coin is successfully spent, the event *spend* will be emitted without any matching event *withdraw*, violating the property. Similarly, in the case of a successful double spending the event *spend* will be emitted twice, but these events are preceded by only one occurrence of the event *withdraw*.

In the rest of the paper, we illustrate all our notions with the "*real cash*" system (mainly coins and banknotes) as a running example. We hope that it helps the reader to understand the properties but also to feel the difference between real cash and e-cash systems.

Example 1. (Real Cash). *In real cash, unforgeability is ensured by physical measures that make forging or copying coins and banknotes difficult, for example by adding serial numbers, using special paper, ultraviolet ink, holograms and so on.*

Since a malicious client might be interested to create fake coins or double spend a coin, it is particularly interesting to study *Unforgeability* with an honest bank and corrupted clients. A partially corrupted seller, which *e.g.,* gives some information to the attacker but still emits the event *spend* correctly, could also be considered to check if a seller colluding with the client and the attacker can results in a coin forging. Note that if the seller is totally corrupted then *Unforgeability* will be trivially violated, since a corrupted seller can simply emit the event *spend* for a forged coin, although there was no transaction.

In case of double spending, the bank should be able to identify the responsible client. This is ensured by *Double Spending Identification*, which says that a client cannot double spend a coin without revealing his identity.

To deposit a coin at the bank the seller has to present a *transaction* which contains, in addition to the coin, some information certifying that he received the coin in a payment. A *valid transaction* is a transaction which could be accepted by the bank, *i.e.,* it contains a correct proof that the coin is received in a correct payment. The bank accepts a valid transaction if it does not contain a coin that is already deposited using the same or a different transaction.

In the following, we denote by $\mathtt{TR}$ the set of all transactions, and we define the function $\mathtt{transId}$ which takes a transaction $tr \in \mathtt{TR}$ and returns a pair (s,c), where s identifies tr and c is the coin involved in tr. Such a pair can usually be computed from a transaction. We also denote by $\mathtt{ID}$ the set of all client identities, and by $\mathtt{D}$ a special data set that includes the data known to the bank after the protocol execution, *e.g.,* the data presents in the bank's database.

Definition 4. **(Double Spending Identification).** *An e-cash protocol ensures* Double Spending Identification *if there exists a test* $\mathtt{T_{DSI}} : \mathtt{TR} \times \mathtt{TR} \times \mathtt{D} \mapsto \mathtt{ID} \cup \{\perp\}$ *satisfying: for any two valid transactions* tr_1 *and* tr_2 *that are different but involve the same coin (*i.e., $\mathtt{transId}(tr_1) = (s_1, c)$*, and* $\mathtt{transId}(tr_2) = (s_2, c)$ *for some coin* c *with* $s_1 \neq s_2$*), there exists* $p \in \mathtt{D}$ *such that* $\mathtt{T_{DSI}}(tr_1, tr_2, p)$ *outputs* $(idc, e) \in \mathtt{ID} \times \mathtt{D}$*, where* e *is an evidence that* idc *withdrew the coin* c*.*

Double Spending Identification allows the bank to identify the double spender by running a test $\mathtt{T_{DSI}}$ on two different transactions that involves the same coin. For example, consider a protocol where after a successful transaction the seller gets $x = m.id + r$ where id is the identity of the client (*e.g.,* his secret key), r is a random value (identifies the coin) chosen by the client at withdrawal, and m is the challenge of the seller. So, if the client double spends the same coin then the bank can compute id and r using the two equations: $x_1 = m_1.id + r$ and $x_2 = m_2.id + r$. The data p could be some information necessary to identify the double spender or to construct the evidence e. This data is usually presented to the bank at withdrawal or at deposit. The required evidence depends on the protocol. Note that e is an evidence from the point of view of the bank, and not necessarily a proof for an outer judge. Thus, the goal of *Double Spending Identification* is to preserve the security of the bank so that he can detect and identify the responsible of double spending when happens. Note that, if a client withdraws a coin and gives it to an attacker which double spends it, then the test returns the identity of the client and not the attacker's identity.

Example 2. **(Real Cash).** *In real cash, double spending is prevented by ensuring that notes cannot be copied. However,* Double Spending Identification *is not ensured: even if a central bank is able to identify copied banknotes using,* e.g.*, their serial numbers, this does not allow it to identify the person responsible for creating the counterfeit notes.*

Double Spending Identification gives rise to a potential problem: what if the client is honest and spends the coin only once, but the attacker (*e.g.,* a corrupted seller) is able to forge a second spend, or what if a corrupted bank is able to simulate a coin withdrawal and payment *i.e.,* to forge a coin withdrawal and payment that seems to be made by a certain client. For instance, in the example mentioned above, the two equations are enough evidence for the bank. However, if the bank knows id he can generate the two equations himself and blame the client for double spending. So, to convince a judge, an additional evidence is needed, *e.g.,* the client's signature.

If any of the two situations mentioned above is possible, then a honest client could be falsely blamed for double spending, and also it gives raise to a corrupted client which is responsible of double spending to deny it. To solve this problem we define *Exculpability*, which says that the attacker, even when colluding with the bank and the seller, cannot forge a double spend by a certain client in order to blame him. More precisely, provided a transaction executed by a client idc, the attacker cannot provide two different valid transactions which involves the same coin, and the data p necessary for the test $\mathtt{T_{DSI}}$ to output the identity idc with an evidence. Note that *Exculpability* is only relevant if *Double Spending Identification* holds: otherwise a client cannot be blamed regardless of the ability to forge a second spend or to simulate a coin withdrawal and payment, as his identity cannot be revealed.

Definition 5. **(Exculpability).** *Assume that we have a test* $\mathtt{T_{DSI}}$ *as specified in Def. 4,* i.e., Double Spending Identification *holds, and that the bank is corrupted. Let* idc *be a honest client (in particular he does not double spend a coin), and* ids *be a corrupted seller. Then,* Exculpability *is ensured if, after observing a* *transaction made by idc with ids, the attacker cannot provide two valid transactions $tr_1, tr_2 \in \mathtt{T}$ that are different but involve the same coin c, and some data p such that $\mathtt{T}_{\mathtt{DSI}}(tr_1, tr_2, p)$ outputs (idc, e) where e is an evidence that idc withdrew the coin c.*

The intuition is: if the attacker can provide two transactions tr_1, tr_2 such that $\mathtt{T}_{\mathtt{DSI}}(tr_1, tr_2)$ returns a client's identity (that is, the two different valid transactions involve same coin), then it was able to forge (at least) one transaction since the honest client performs (at most) one transaction per coin.

If after observing a transaction executed by a client *idc*, the attacker can provide a different valid transaction which involves the same coin, and the required data *p*, then the test will return the identity *idc* with the necessary evidence, thus the property will be violated. Similarly, in the case where the attacker can forge a coin withdrawal and payment seems to be made by a client *idc*, then the attacker can obtain two transactions satisfying the required conditions, together with the necessary data *p*, so that the test will return the identity *idc* with an evidence.

Note that, *Double Spending Identification* and *Exculpability* are only relevant in case of off-line e-cash systems where double spending might be possible.

Example 3. (Real Cash). *As* Double Spending Identification *is not ensured in real cash, exculpability is not relevant: any client that posses a counterfeit banknote can plausibly deny that he produced this note.*

3.2 Privacy Properties

We express our privacy properties as observational equivalence, a standard choice for such kind of properties. We use the *labeled bisimilarity* ($\approx_l$) to express the equivalence between two processes (AF01). Informally, two processes are equivalent if an attacker interacting with them observer has no way to tell them apart.

To ensure the privacy of the client, the following two notions have been introduced by cryptographers and are standard in the literature e.g., (CG08; Fer94; Sch97).

1. *Weak Anonymity*: the attacker cannot link a client to a spend, *i.e.,* he cannot distinguish which client makes the payment.
2. *Strong Anonymity*: additionally to weak anonymity, the attacker should not be able to decide if two spends were done by the same client, or not.

In (CG08), *Weak Anonymity* is defined as the following game: two honest clients each withdraw a coin from the bank. Then one of them (randomly chosen) spends his coin to the adversary. The adversary already knows the identities of these two clients, and also the secret key of the bank. It wins the game if it guesses correctly which client spends the coin. Inspired by this definition, we define *Weak Anonymity* in the applied π-calculus as follows:

Definition 6. (Weak Anonymity). *An e-cash protocol ensures* Weak Anonymity *if for any e-cash instance CP, any two honest clients idc_1, idc_2, any corrupted seller ids, we have that: $CP_I[C\sigma_{idc_1}\sigma_{c_1}\sigma_{ids}|C_w\sigma_{idc_2}\sigma_{c_2}|S^c\sigma_{ids}|B^c] \approx_l CP_I[C_w\sigma_{idc_1}\sigma_{c_1}|C\sigma_{idc_2}\sigma_{c_2}\sigma_{ids}|S^c\sigma_{ids}|B^c]$, where c_1, c_2 are any two coins (not previously known to the attacker) withdrawn by idc_1 and idc_2 respectively, $I = \{idc_1, idc_2, ids, id_B\}$, id_B is the bank's identity, and C_w is a variant of C that halts at the end of the withdrawal phase.*

Weak anonymity ensures that a process in which the client idc_1 spends the coin c_1 to the corrupted seller ids_1, is equivalent to a process in which the client idc_2 spends the coin c_2 to the corrupted seller ids_1. We assume a corrupted bank represented by B^c. Note that the client that does not spend his coin still withdraws it. This is necessary since otherwise the attacker could likely distinguish both sides during the withdrawal phase, as the bank is corrupted and typically the client reveals his identity to the bank at withdrawal so that his account can be charged. We also note that we do not necessarily consider other corrupted clients, however this can easily be done by replacing some honest clients from the context CP_I (*i.e.,* other than idc_1 and idc_2) with corrupted ones.

Example 4. (Real Cash). *Real coins ensure weak anonymity as two coins (assuming the same value and production year) are indistinguishable. However, banknotes do not ensure weak anonymity according to our definition, as they include serial numbers. Since the two clients withdraw a note each, the notes hence have different serial numbers which the bank can identify. In reality this is used by central banks to trace notes and detect suspicious activities that,* e.g., *could hint at money laundering. Note however that banknotes ensure a weaker form of anonymity: if two different clients use the same note, one cannot distinguish them.*

Strong Anonymity is defined in (CG08) using the same game as for *Weak Anonymity*, with the difference that the adversary may have previously seen some coins being spent by the two honest clients explicitly mentioned in the definition. We define *Strong Anonymity* as follows:

Definition 7. (Strong Anonymity.) *An e-cash protocol ensures* Strong Anonymity *if for any e-cash in-*

stance CP, any two honest clients idc_1, idc_2, any corrupted seller ids, we have that:

$$CP_I[|_{0\leq i\leq m_1} C\sigma_{idc_1}\sigma_{c_1^i}\sigma_{ids}|_{0\leq i\leq m_2} C\sigma_{idc_2}\sigma_{c_2^i}\sigma_{ids}|$$

$$C\sigma_{idc_1}\sigma_{c_1}\sigma_{ids}|C_w\sigma_{idc_2}\sigma_{c_2}|S^c\sigma_{ids}|B^c] \approx_l$$

$$CP_I[|_{0\leq i\leq m_1} C\sigma_{idc_1}\sigma_{c_1^i}\sigma_{ids}|_{0\leq i\leq m_2} C\sigma_{idc_2}\sigma_{c_2^i}\sigma_{ids}|$$

$$C_w\sigma_{idc_1}\sigma_{c_1}|C\sigma_{idc_2}\sigma_{c_2}\sigma_{ids}|S^c\sigma_{ids}|B^c]$$

where c_1 and $c_1^1 \dots c_1^{m_1}$ are any coins withdrawn by idc_1, c_2 and $c_2^1 \dots c_2^{m_2}$ are any coins withdrawn by idc_2, $I = \{idc_1, idc_2, ids, id_B\}$, id_B is the bank's identity, and C_w is a variant of C that halts at the end of the withdrawal phase.

Strong Anonymity ensures that the process in which the client idc_1 spends $m_1 + 1$ coins, while idc_2 spends m_2 coins and additionally withdraws another coin without spending it, is equivalent to the process in which the client idc_1 spends m_1 coins and withdraws an additional coin, while idc_2 spends $m_2 + 1$ coins. The definition assumes that the bank is corrupted, and that the seller receiving the coins from the two clients idc_1 and idc_2 is also corrupted. Note that, we consider C_w to avoid distinguishing from the number of withdrawals by each client.

Again, we can replace some honest clients from CP_I by corrupted ones.

Example 5. (Real Cash). *Again, real coins ensure strong anonymity as, assuming the same value and production year, two coins are indistinguishable. Yet, for the same reason as in weak anonymity, banknotes do not ensure strong anonymity according to our definition: the serial numbers allow an attacker to identify the different clients.*

We note that any protocol satisfying Strong Anonymity also satisfies Weak Anonymity, as Weak Anonymity is a special case of Strong Anonymity for $m_1 = m_2 = 0$, *i.e.* when the two honest clients do not make any previous spends.

4 CASE STUDY: CHAUM'S PROTOCOL

David Chaum proposed the first (on-line) e-cash system in (Cha83) based on blind signatures, and an off-line variant of the protocol is proposed in (CFN90). A real implementation based on these two variants, allowing users to make purchases over open networks such as the Internet, was put in service by DigiCash Inc. The corporation declared bankruptcy in 1998, and was sold to Blucora[3] (formerly Infospace Inc.). The on-line protocol implemented by DigiCash is presented in (Sch97).

In the following, we describe and analyze both the on-line and the off-line variants of the protocol, as well as, the on-line protocol implemented by DigiCash. For this we use ProVerif an automatic tool that verifies cryptographic protocols (Bla01). All the verification presented in the paper are carried out on a standard PC (Intel(R) Pentium(R) D CPU 3.00GHz, 2GB RAM).

4.1 Chaum's On-line Protocol

The Chaum On-line Protocol was proposed in (Cha83) and detailed in (CFN90). It allows a client to withdraw a coin blindly from the bank, and then spend it later in a payment without being traced even by the bank. The protocol is "on-line" in the sense that the seller does not accept the payment before contacting the bank to verify that the coin has not been deposited before, to prevent double spending. We start by giving a description of the protocol.

Withdrawal Phase: To obtain an electronic coin, the client communicates with the bank using the following protocol:

1. The client randomly chooses a value x, and a coefficient r, the client then sends to the bank his identity u and the value $b = \mathtt{blind}(x, r)$, where `blind` is a blinding function.
2. The bank signs the blinded value b using a signing function `sign` and his secret key skB, then sends the signature $bs = \mathtt{sign}(b, skB)$ to the client. The bank also debits the amount of the coin from the client's account.
3. The client verifies the signature and removes the blinding to obtain the bank's signature $s = \mathtt{sign}(x, skB)$ on x. The coin consists of the pair $(x, \mathtt{sign}(x, sk_B))$.

Payment (and Deposit) Phases: To spend the coin

1. The client sends the pair $(x, \mathtt{sign}(x, sk_B))$ to the seller.
2. After checking the bank's signature, the seller sends the coin $(x, \mathtt{sign}(x, sk_B))$ to the bank to verify that it is not deposited before.
3. The bank verifies the signature s, and that the coin is not in the list of deposited coins. If these checks succeed the bank credits the seller's account with the amount of the coin and informs him of acceptance. Otherwise, the payment is rejected.

[3] http://www.blucora.com/

Modeling in ProVerif: We use ProVerif to perform the automatic protocol verification. ProVerif uses a process description based on the applied π-Calculus, but has syntactical extensions and is enriched by events to check reachability and correspondence properties. Besides, it can check equivalence properties. As explained above, we model privacy properties as equivalence properties, and we use events to verify the other properties.

The equational theory depicted in Table 1 models the cryptographic primitives used within Chaum online protocol. It includes well-known model for digital signature (functions *sign*, *getmess*, and *checksign*). The functions *blind/unblind* are used to blind/unblind a message using a random value. We also include the possibility of unblinding a signed blinded message, so that we obtain the signature of the message – the key feature of blind signatures.

Table 1: Equational theory.

$$getmess(sign(m,k)) = m$$
$$checksign(sign(m,k), pk(k)) = m$$
$$unblind(blind(m,r),r) = m$$
$$unblind(sign(blind(m,r),k),r) = sign(m,k)$$

Analysis: The result of the analysis is summarized in Table 2.

We model *Unforgeability* as an injective correspondence between the two events *withdraw* and *spend*, they are placed in their appropriate positions, according to the Def. 3, inside the bank and seller processes respectively. We consider a honest bank and honest seller but corrupted clients. We assume that the bank sends an authenticated message through private channel to inform the seller about a coin acceptance. Otherwise, the attacker can forge a message which leads the seller to accepting an already deposited coin. However, ProVerif still finds an attack against *Unforgeability* when two copies of the same coin spent at the same time. In this case the bank makes two parallel database lookups to check if the coin was deposited before. If the parallel deposit was not finished yet and thus the coin is not yet inserted in the database, then each lookup confirms that the coin was not deposited before which results in acceptance of two spends of the same coin. This attack may be avoided with some synchronization like locking the table when a coin deposit is initiated and then unlocking it when the operation is finished. ProVerif does not support such an feature. Protocols that rely on state could be analyzed using the Tamarin Prover[4] thanks to the SAPIC[5] tool (we keep this for future work).

Table 2: Analysis of the Chaum on-line protocol. A ✓ indicates that the property holds. A × indicates that it fails (ProVerif shows an attack).

Property	Result	Time
Unforgeability	×	$< 1s$
Weak Anonymity	✓	$< 1s$
Strong Anonymity	✓	$< 1s$

Note that corrupted clients cannot create a fake coin as the correspondence holds without injectivity.

Double Spending Identification and *Exculpability* are not relevant in the case of on-line protocols as their countermeasure against double spending is the on-line calling of the bank at payment, and thus they do not have any kind of test to identify double spenders.

For privacy properties, we assume a corrupted bank and a corrupted seller, but honest clients. ProVerif confirms that the privacy of the client is preserved, as both *Weak Anonymity*, and *Strong Anonymity* are satisfied. This due to the fact that the coin is signed blindly during the withdrawal phase, and thus cannot be traced later by the attacker even when colludes with the bank and the seller. Note that, for *Strong Anonymity*, we consider an unbounded number of spends by each client and one spend that is made by either the first client or by the second one.

4.2 DigiCash On-line Protocol

The on-line protocol implemented by DigiCash Inc. is outlined in (Sch97). It has the same withdrawal phase as Chaum on-line protocol, except that the client sends an authenticated coin to be signed by the bank, however the paper does not specify the way of authentication. We ignore this authentication as its purpose is to ensure that the bank debits the correct client account. Hence, we believe that it does not effect the privacy and unforgeability properties (analysis confirms that as we can see in Table 2). The payment and deposit phases are different from those of Chaum on-line protocol. They are summarized as follows:

Payment (and Deposit) Phases in DigiCash:

1. The client sends to the seller $pay = enc((id_s, \mathtt{h}(pay\text{-}spec), x, \mathtt{sign}(x, sk_B)), pk_B)$ which is the encryption, using the public key of the bank pk_B, of the seller's identity id_s, hash of

[4] http://www.infsec.ethz.ch/research/software/tamarin.html

[5] http://sapic.gforge.inria.fr/

Table 3: Analysis of DigiCash on-line protocol. A ✓ indicates that the property holds. A × indicates that it fails (ProVerif shows an attack).

Property	Result	Time
Unforgeability	×	$< 1s$
Weak Anonymity	✓	$< 1s$
Strong Anonymity	✓	$< 1s$

the payment specification *pay-spec* (specification of the sold object, price etc), and the coin $(x, \texttt{sign}(x, sk_B))$.

2. The seller signs $(\texttt{h}(\textit{pay-spec}), \textit{pay})$ and sends it along with his identity id_s to the bank.
3. The bank verifies the signature, decrypts *pay* then verifies the value of $\texttt{h}(\textit{pay-spec})$ and that the coin is valid and not deposited before. If so it informs the seller to accept the coin, and to reject it otherwise.

Modeling in ProVerif: Additionally to the equational theory of the Chaum on-line protocol (Table 1), the equational theory of DigiCash on-line protocol includes well-known model of the public key encryption represented by the following equation: $dec(enc(m, pk(k)), k) = m$.

Analysis: The result of analysis of DigiCash online protocol using ProVerif is summarized in Table 3. ProVerif shows the same results as obtained for Chaum on-line protocol. Namely, it shows that *Weak Anonymity*, and *Strong Anonymity* are satisfied, and it outputs the same attack presented in Section 4.1 against *Unforgeability*. Again *Double Spending Identification* and *Exculpability* are not relevant.

Note that, obtaining the same result for the two protocols, even that they have different payment and deposit phases, confirms that the blinding signature used during the withdrawal phase plays the key role in preserving the privacy of the client, as claimed by David Chaum.

4.3 Chaum's Off-line Protocol

The off-line variant of the Chaum protocol is proposed in (CFN90). It removes the requirement that the seller must contact the bank during every payment. This introduces the risk of double spending a coin by a client.

Withdrawal Phase: to obtain an electronic coin, the client randomly chooses a, c and d, and calculates the pair $H = (\texttt{h}(a, c), \texttt{h}(a \oplus u, d))$, where u is the client identity and $\texttt{h}$ is a hash function. The client then proceed as in the Chaum on-line protocol but with x (the potential coin) replaced by the pair H. Namely, the client blinds the pair H and sends it to the bank. Then the bank signs and returns it to the client. The main difference from the Chaum on-line protocol is that the coin has to be of the following form

$$(\texttt{h}(a, c), \texttt{h}(a \oplus u, d))$$

where the client identity is masked inside it. This aims to reveal the identity if the client later double spends the coin. In order for the bank to be sure that the client provides a message of the appropriate form, Chaum *et al.* used in (CFN90) the well known "*cut-and-choose*" technique. Precisely, the client computes n such a pair H where n is the system security parameter. The bank then selects half of them and asks the client to reveal their corresponding parameters (a, b, c and r). If n is large enough the client can cheats with a low probability.

At the end of this phase the client holds the electronic coin composed of the pair H, and the bank's signature $S = \texttt{sign}(H, sk_B)$. The client also has to keep the random values a, c, d which are used later to spend the coin.

Payment Phase:

1. To make the payment, the client presents the pair H and the bank's signature S to the seller. The seller checks the signature, if it is correct then he chooses and sends a random binary bit y, a challenge, to the client. The client returns to the seller:
   - The values a and c if y is 0.
   - The values $a \oplus u$ and d if y is 1.
2. The seller checks the compliance of the values sent with the pair H. If everything (the signature and the values) is correct, the payment is accepted.

At the end of the payment phase, the seller holds the pair H, the signature S, the values of either (a, c) or $(a \oplus u, d)$, and the challenge y. All these data together compose the transaction the seller has to present to the bank at deposit.

Note, in case where n pairs are used for the coin, the challenge y will be n bit string and for each bit either the corresponding values of (a, c) or $(a \oplus u, d)$ are revealed to the seller.

Deposit Phase:

1. The seller contacts the bank and provides it with the transaction (H, S, y, (a, c)) or (H, S, y, $(a \oplus u, d)$).

2. The bank checks the signature and also whether the values (a,c) or $(a \oplus u,d)$ correspond to their hash value in H. If any of these values is incorrect, the fault is on the seller's part, as he was able to independently check the regularity of the coin at payment. If the coin is correct, the bank checks its database to see whether the same coin had been used before. If it has not, the bank credits the seller's account with the appropriate amount. Otherwise, the bank rejects the transaction.

Chaum off-line protocol does not prevent double spending, however it preserve client's anonymity only if he spend a coin once.

Note that, a double spender can be identified when the coin has the form $(\mathtt{h}(a,c),\mathtt{h}(a \oplus u,d))$. However, the bank can simulate the coin withdrawal and payment (as the bank knows the identities of all the clients), thus the bank can blame a honest client for double spending. As a countermeasure, the authors propose to concatenate two values z and z' with u inside the pair H to have $(\mathtt{h}(a,c),\mathtt{h}(a \oplus (u,z,z'),d))$ and provide to the bank, at withdrawal, additionally the client's signature on $\mathtt{h}(z,z')$.

Modeling in ProVerif: To mode the Chaum off-line protocol in ProVerif, in addition to the equational theory used for the Chaum on-line protocol (Table 1), we use the function *xor* to represent the exclusive or ($\oplus$) of two values. Given the first value, the second value can be obtained using the function *unxor*. Such an – admittedly limited – modeling for $\oplus$ operator is sufficient to catch the functional properties of the scheme required by Chaum off-line protocol, but does not catch all algebraic properties of this operator. However, there are currently no tools that support observational equivalence – which we need for the anonymity properties – and all algebraic properties of $\oplus$. Kuesters *et al.* (KT11) proposed a way to extend ProVerif with $\oplus$. Their tool translates a model of the protocol to a ProVerif input where all $\oplus$ are ground terms to enable automated reasoning. However, this tool can only deal with secrecy and authentication properties, and does not support equivalence properties. The *xor* function is only used to hide the client's identity u using a random value a $(a \oplus u)$, which we model as $xor(a,u)$. The bank then uses a to reveal the client's identity u if he double spends a coin. This is modeled by the following two equations

$$unxor(xor(a,u),a) = u$$
$$unxor(a,xor(a,u)) = u$$

which represents the various ways: $((a \oplus u) \oplus a) = u$, or $(a \oplus (a \oplus u)) = u$. We always assume that identity is the second value, and this is how we model it inside honest processes.

Analysis: As expected ProVerif confirms that *Unforgeability* is not satisfied, a corrupted client can double spend a coin. In fact the seller cannot know whether a certain coin is already spent or not, he accepts any coin that is certified by the bank. However, a collusion between the client and the attacker cannot lead to forging a coin.

In case of double spending, the bank may receive two transactions of the form $tr_1 = (h,hx,sign((h,hx),skB),0,a,c)$ and $tr_1 = (h,hx,sign((h,hx),skB),1,xor(a,u),d)$. The bank can apply a test to obtain the identity u. This is done using the *unxor* function as $unxor(xor(a,u),a) = u$. The evidence here is showing that the identity of the client is masked inside the coin. This can be done thanks to the values of $(a,c,xor(a,u),d)$ which are initially known only to the client. Spending the coin only once reveals either (a,c) or $(xor(a,u),d)$ which does not allow to obtain the identity u. Note that, if the two sellers provide the same challenge, the two transactions will be exactly equal. In this case no double spending is detected, and the second transaction will be rejected by the bank which considers it as a second copy of the first transaction. In practice this can be avoided with high probability if n pairs coin is used and thus n bits challenge. Note that ProVerif consider all the possibilities.

We model the output of an identity and an evidence of the test $\mathtt{T_{DSI}}$ by an emission of the *event OK*, and *event KO* otherwise. To say that *Double Spending Identification* is satisfied we should have that the test $\mathtt{T_{DSI}}$ does not emit the *event KO* for every two valid transactions tr_1, tr_2 that are different but involves the same coin, *i.e.,* it always emits *event OK* for such transactions. ProVerif shows that the test can emit the *event KO* for certain two transactions satisfy the required conditions. Actually, a corrupted client can withdraw a coin that does not have the appropriate form (*e.g.,* client's identity is not masked inside it), thus the bank cannot obtain the identity in case of double spending. Note that, if the bank only certifies coins with the appropriate form at withdrawal (*i.e.,* of the form $(\mathtt{h}(a,c),\mathtt{h}(a \oplus u,d))$), then the property holds, ProVerif confirms that. Again, in practice applying the "cut-and-choose" technique can guarantee with high probability that the coin is in the appropriate form. However, applying this technique using Proverif does not make any difference since ProVerif works under symbolic world which deals with possibilities and not with probabilities. For instance, the

Table 4: Analysis of Chaum off-line protocol. A ✓ indicates that the property holds. A × indicates that it fails (ProVerif shows an attack).
(∗) Only coins with the appropriate form are considered.
(†) After applying the countermeasure.

Property	Result	Time
Unforgeability	×	<1s
Double Spending Identif.	×	<2s
Double Spending Identif.*	✓	<2s
Exculpability*	×	< 6s
Exculpability†	✓	< 6s
Weak Anonymity	✓	<1s
Strong Anonymity	✓	<1s

attacker still can guess the pairs that the bank will request to reveal and construct them in the appropriate form, but cheat with the others which will compose the coin.

We analyze *Exculpability* in case where only coins of appropriate forms are considered *i.e.,* the case where *Double Spending Identification* holds. ProVerif confirms that a corrupted bank can blame a honest client. The bank can simulate the withdrawal and the payment since the bank knows the identity of the client. Thus it can obtain two transactions satisfying the required conditions. This is due to the fact that the evidence obtained by the test, which is showing that the client's identity is masked inside the coin, is not strong enough to act as a proof. However, the attacker cannot re-spend a coin withdrawn and spent by a honest client.

After applying the countermeasure that is including some terms z and z' so that the client signs $\mathtt{h}(z,z')$, ProVerif confirms that *Exculpability* holds. Applying the countermeasure results in a new test which takes, in addition to the two transactions, the client's signature on $\mathtt{h}(z,z')$. The test shows, in case of double spending, that the identity u and the preimage (z,z') of the hash signed by the client are masked inside the coin. This represents a stronger evidence which acts as a proof that the client withdrew the coin since the bank cannot forge the client's signature.

We note that, Ogiela *et al.* (OS14) show an attack on Chaum's off-line protocol: when a client double spends a coin, the sellers can forge additional transactions involving the same coin, so that the bank cannot know how many transactions are actually result from spends made by the client and how many are forged by the sellers. In such a case, according to our definition, *Unforgeability* does not hold since the client has to spend the coin at least twice. Yet, corrupted sellers can blame a corrupted client who double spends a coin for further spends. Moreover the bank can still identify the client and punish him as the bank can be sure that he at least spend the coin twice.

Concerning privacy properties, ProVerif shows that Chaum off-line protocol still satisfies both *Weak Anonymity* and *Strong Anonymity*.

To sum up, ProVerif confirms the claim about preserving client's anonymity. ProVerif also was able to show that a client can double spend a withdrawn coin but cannot forge a coin, and that the bank can identify the double spender if the coin is in the appropriate form. ProVerif also shows, in case of coin with appropriate form, that the bank can simulate a withdrawal and payment, and thus can blame him for double spending. After applying the countermeasure no attack against *Exculpability* is found.

5 CONCLUSIONS

E-cash protocols can offer anonymous electronic payment services. Numerous protocols have been proposed in the literature, and multiple flaws were discovered. To avoid further bad surprises, formal verification can be used to improve confidence in e-cash protocols. In this paper, we developed a formal framework to automatically verify e-cash protocols with respect to multiple essential privacy and forgery properties. Our framework relies on the applied π-calculus and uses ProVerif as the verification tool. As a case study, we analyzed the on-line protocol proposed by Chaum *et al.* , as well as, a real implementation based on it. We also analyze the off-line variant of this system. We confirm some claims and known weaknesses. We also identified that some synchronization is necessary in case of on-line protocols to prevent double spending.

As future work, we would like to investigate further case studies and to extend our model to cover transferable protocols with divisible coins. Also we would like to use the tool SAPIC based on Tamarin, in order to see how it can help to analyze e-cash protocols.

REFERENCES

Sattar J. Aboud and Ammar Agoun. Analysis of a known offline e-coin system. *International Journal of Computer Applications*, 2014.

Masayuki Abe and Eiichiro Fujisaki. How to date blind signatures. In *Advances in Cryptology - ASIACRYPT '96, Korea, November 3-7, 1996, Proceedings*, volume 1163, pages 244–251. Springer, 1996.

Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *The 28th Sym-*

posium on Principles of Programming Languages, ACM, UK, 2001.

M. Backes, C. Hritcu, and M. Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *CSF*, 2008.

Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14), Canada*, 2001.

Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '93, pages 302–318, London, UK, UK, 1994. Springer-Verlag.

David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *Advances in Cryptology: Proceedings of CRYPTO '88*, pages 319–327. Springer New York, 1990.

Sébastien Canard and Aline Gouget. Anonymity in transferable e-cash. In *Applied Cryptography and Network Security, ACNS, USA*, pages 207–223, 2008.

Sébastien Canard, Aline Gouget, and Jacques Traoré. Improvement of efficiency in (unconditional) anonymous transferable e-cash. In *Financial Cryptography and Data Security, 12th International Conference, FC, Mexico*. Springer, 2008.

David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of CRYPTO '82*. Springer US, 1983.

Giovanni Di Crescenzo. A non-interactive electronic cash system. In *Algorithms and Complexity, Second Italian Conference, Italy*, volume 778 of *Lecture Notes in Computer Science*, pages 109–124. Springer, 1994.

Chang Yu Cheng, Jasmy Yunus, and Kamaruzzaman Seman. Estimations on the security aspect of brand's electronic cash scheme. In *19th International Conference on Advanced Information Networking and Applications AINA, Taiwan*, 2005.

I. B. Damgård. Payment systems and credential mechanisms with provable security against abuse by individuals. In *Proceedings on Advances in Cryptology*, pages 328–335. Springer-Verlag, 1990.

Stefano D'Amiano and Giovanni Di Crescenzo. Methodology for digital money based on general cryptographic tools. In *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Italy*. Springer, 1994.

Jannik Dreier, Rosario Giustolisi, Ali Kassem, Pascal Lafourcade, Gabriele Lenzini, and Peter Y. A. Ryan. Formal analysis of electronic exams. In *SECRYPT, Austria, 2014*, pages 101–112, 2014.

S. Delaune, S. Kremer, and M.D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, jul 2009.

J. Dreier, P. Lafourcade, and Y. Lakhnech. A formal taxonomy of privacy in voting protocols. In *ICC*, pages 6710–6715, 2012.

Jannik Dreier, Pascal Lafourcade, and Yassine Lakhnech. Formal verification of e-auction protocols. In *Principles of Security and Trust, POST*, pages 247–266. Springer, 2013.

D. Dolev and Andrew C. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983.

Niels Ferguson. Single term off-line coins. In *Advances in Cryptology, Lecture Notes in Computer Science - EUROCRYPT '93*, volume 765, pages 318–328. Springer-Verlag, 1994.

Chun-I Fan, Vincent Shi-Ming Huang, and Yao-Chun Yu. User efficient recoverable off-line e-cash scheme with fast anonymity revoking. *Mathematical and Computer Modelling*, 2013.

Sangjin Kim and Heekuck Oh. Making electronic refunds reusable, 2001.

Ralf Küsters and Tomasz Truderung. Reducing protocol analysis with xor to the xor-free case in the horn theory based approach. *Journal of Automated Reasoning*, 2011.

Zhengqin Luo, Xiaojuan Cai, Jun Pang, and Yuxin Deng. Analyzing an electronic cash protocol using applied pi calculus. In *Applied Cryptography and Network Security, 5th International Conference, ACNS, China*, 2007.

Tatsuaki Okamoto and Kazuo Ohta. Disposable zero-knowledge authentications and their applications to untraceable electronic cash. In *Proceedings on Advances in Cryptology*, CRYPTO '89, pages 481–496. Springer-Verlag New York, Inc., 1989.

Marek R. Ogiela and Piotr Sulkowski. Improved cryptographic protocol for digital coin exchange. In *Soft Computing and Intelligent Systems (SCIS)*, pages 1148–1151, 2014.

Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. How to break another provably secure payment system. In *EUROCRYPT '95, International Conference on the Theory and Application of Cryptographic Techniques, France*, pages 121–132, 1995.

Birgit Pfitzmann and Michael Waidner. How to break and repair A "provably secure" untraceable payment system. In *CRYPTO '91, 11th Annual International Cryptology Conference, USA*, pages 338–350, 1991.

Berry Schoenmakers. Basic security of the ecash payment system. In *In Applied Cryptography, Course on Computer Security and Industrial Cryptography*, pages 201–231. Springer-Verlag, LNCS, 1997.

Aye Thandar Swe and Khin Khat Khat Kyaw. Formal analysis of secure e-cash transaction protocol. In *International Conference on Advances in Engineering and Technology*, Singapore, 2014.