Automated Verification of E-Cash Protocols

Jannik Dreier, Ali Kassem, Pascal Lafourcade

▶ **To cite this version:**

Jannik Dreier, Ali Kassem, Pascal Lafourcade. Automated Verification of E-Cash Protocols. ICETE 2015 - 12th International Joint Conference on e-Business and Telecommunications, Jul 2015, Colmar, France. 10.1007/978-3-319-30222-5_11 . hal-01840596

HAL Id: hal-01840596
https://hal.science/hal-01840596v1

Submitted on 16 Jul 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Automated Verification of E-Cash Protocols*

Jannik Dreier[1,2,3], Ali Kassem[4,5], and Pascal Lafourcade[6]

[1] Inria, F-54600 Villers-lés-Nancy, France;
[2] CNRS, Loria, UMR 7503, F-54506 Vandoeuvre-lés-Nancy, France
[3] Université de Lorraine, Loria, UMR 7503, F-54506 Vandoeuvre-lés-Nancy, France
[4] Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble, France
[5] Ascola team (Mines Nantes, Inria, Lina) DAPI, École des Mines de Nantes, France
[6] University Clermont Auvergne, LIMOS, France
jannik.dreier@loria.fr, ali.kassem@mines-nantes.fr,
pascal.lafourcade@udamail.fr

Abstract. *Electronic cash (e-cash) permits secure e-payments by providing security and anonymity similar to real cash. Several protocols have been proposed to meet security and anonymity properties of e-cash. However, there are no general formal definitions that allow the automatic verification of e-cash protocols. In this paper, we propose a formal framework to define and verify security properties of e-cash protocols. To this end, we model e-cash protocols in the applied π-calculus, and we formally define five relevant security properties. Finally, we validate our framework by analyzing, using the automatic tool ProVerif, four e-cash protocols: the online and the offline Chaum protocols, the Digicash protocol, and the protocol by Petersen and Poupard.*

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

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 all details of the clients' transactions. By analyzing a clients payments for, *e.g.*, transportation, 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 concept of "untraceable" e-cash was introduced by David

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

Chaum [Cha83]. The general e-cash system involves three main parties: client, bank and seller. A client withdraws an electronic coin from the bank, which blindly signs it. The bank is the only party capable to create coins. The client then can use the coins to pay a seller. Finally, the seller deposits the coin he received from client in his bank account. At deposit the bank verifies that the coin was not deposited before. If the coin was deposited before, the banks verifies whether the seller deposited the same coin twice, or the client double-spent the coin. In the former case the bank reject the deposit, while in the latter case the bank takes actions depending on the type of the system (online or offline). In online e-cash systems, *i.e.*, those where a seller has to contact the bank at payment before accepting the coin, the bank acknowledge the seller to reject the coin at payment. Thus, in such systems normally double spending is not possible. In contrast, in offline systems the bank runs a procedure to disclose the client's identity. This can be achieved due to the fact that offline systems usually support the encoding of client's identity into the coin at its withdrawal. To protect the client, an e-cash system must also ensures that the bank cannot frame a honest client for double spending. We identify the following main security properties of e-cash protocols:

- *Unforgeability*, which says that clients cannot spend more coins than they withdrew.
- *Double Spending Identification* ensures that the bank can identify the double spender.
- *Exculpability* ensures that an attacker cannot forge a double spend, and hence incorrectly blame an honest client for double spending.
- *Weak Anonymity* ensures that the attacker cannot link a client to a payment.
- *Strong Anonymity* ensures, additionally to weak anonymity, that the attacker cannot decide whether two payments were made by the same client.

Contributions. In this paper, we propose a general formalization for e-cash protocols in the applied π-calculus [AF01]. Our definitions are amenable to automatic verification using ProVerif [Bla01], and cover all the identified unforgery and privacy properties: *Unforgeability*, *Double Spending Identification*, *Exculpability*, *Weak Anonymity*, *Strong Anonymity*. 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 offline variant of this e-cash system [CFN90], and the protocol by Peterson and Poupard [PP97]. Some of the results have been published in a previous paper [DKL15]. This paper extends our results, and provides an additional case study.

Related Work. Several e-cash protocols have been proposed [CFN90, Dam90, DC94, Cre94, Bra94, AF96, KO02, FHY13, PP97] since the seminal work by David Chaum [Cha83], which introduces the blinding signature primitive to allow the anonymous withdrawal of coins. Chaum *et al.* [CFN90] presented an offline variant of Chaum [Cha83] protocol. Berry Schoenmakers has described a real e-cash protocol that is implemented by DigiCash based on online Chaum

protocol [Cha83]. Abe *et al.* [AF96] have introduced a scheme based on partial blind signature, which allows 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.* [KO02] have proposed an e-cash system that supports coin refund and assigns them a value, based again on partial blind signature. Peterson and Poupard [PP97] have proposed a protocol to prevent extortion attacks by relying on additional party, the trustee, which publishes, in case of coin or key extortion, some information that can be used by the sellers to reject illegal coins.

However, several 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 and Agoun [AA14] show that [FHY13] cannot ensure the anonymity and unlinkability properties that were claimed. These numerous attacks triggered some work on formal analysis of e-cash protocols in the computational [CG08] and symbolic world [LCPD07, SK14]. Canard and Gouget [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 (ProVerif [Bla01], and AVISPA [ABB$^+$05] respectively). Yet, they only consider a fraction of the essential security properties, and for some properties Luo *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.

Outline. In Section 2, we model e-cash protocols in the applied π-calculus. Then, in Section 3, we formalize the security properties. In Section 4, we validate our framework by analyzing, using ProVerif [Bla01], the online and the offline e-cash systems by Chaum *et al.* [Cha83, CFN90], the implementation based on the online protocol [Sch97], and the protocol by Peterson and Poupard [PP97]. Finally, in Section 5, we discuss our results and outline future work.

2 Modeling E-cash Protocols in The Applied π-Calculus

An e-cash system involves at least the following parties: the *client* C who has an account at the *bank* B, the *seller* S who accepts electronic coins, and the bank, which certifies the electronic coins. A protocol can has several authorities that run in parallel with the bank. A typically e-cash protocol runs 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 to return coins directly to the bank without using them in a payment, or to restore coins that have been lost. 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 model e-cash protocols in the applied π-calculus. We refer to the original paper [AF01] for a detailed description of its syntax and semantics. We assume 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. 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* $(B, S, C, \tilde{n}_p)$*, 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* $\tilde{n}_p$ *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:*

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

where $\tilde{n}$ *is the set of all restricted names which includes the set of the protocol's private channels* $\tilde{n}_p$*;* 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. Note also that, the bank process B can be structurally equivalent to $B_1, \ldots, B_k$, thus several authorities can be captured by our definition.

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 $CP_{\{idc_1\}}[C\sigma_{idc_1}\sigma_{c_{11}}\sigma_{ids_{11}}| \ldots |C\sigma_{idc_1}\sigma_{c_{1p_1}}\sigma_{ids_{1p_1}}]$.

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 Formalizing Security Properties

In this section, we propose formal definitions for the three forgery-related properties: *Unforgeability*, *Double Spending Identification*, and *Exculpability*, as well as, for the two privacy properties: *Weak Anonymity* and *Strong Anonymity*.

3.1 Forgery-Related Properties

In an e-cash protocol only the bank should be able to create coins. A client must not be able to forge a coin, or to double spend a valid coin. This is ensured by *Unforgeability*, which we define using 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 before 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.

Note that, 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*. 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 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.

In case of double spending, the bank should be able to identify the responsible client. This is ensured by *Double Spending Identification* (in short, DSI), 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 DSI if there exists a test* $\mathtt{T}_{\mathtt{DSI}} : \mathtt{TR} \times \mathtt{TR} \times \mathtt{D} \mapsto \mathtt{ID} \cup \{\perp\}$ *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* $\mathtt{T}_{\mathtt{DSI}}(tr_1, tr_2, p)$ *outputs* $(idc, e) \in \mathtt{ID} \times \mathtt{D}$, *where* e *is an evidence that* idc *withdrew the coin* c.

DSI allows the bank to identify the double spender by running a test $\mathtt{T}_{\mathtt{DSI}}$ 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 DSI is to preserve the security of the bank by enabling him to identify the responsible of a double spending. Note also 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, DSI 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.

DSI 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 $\mathtt{T}_{\mathtt{DSI}}$ to output the identity *idc* with an evidence. Note that *Exculpability* is only relevant if DSI 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* $\mathtt{T}_{\mathtt{DSI}}$ *as specified in Definition 4, i.e. DSI 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 *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 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*, together with the necessary data p. Then the attacker can obtain

two transactions satisfying the required conditions, and 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.

3.2 Privacy Properties

We define the privacy properties using 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 has no way to tell them apart.

To ensure the privacy of the client, the following two notions have been introduced by cryptographers, *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 a payment.
2. *Strong Anonymity*: a stronger notion than weak anonymity, which additionally requires that the attacker cannot decide whether two spends were done by the same client.

Canard *et al.* [CG08], have defined *Weak Anonymity* using 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 a corrupted seller ids_1, is equivalent to a process in which the client idc_2 spends the coin c_2 to a 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. 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 3 (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 the bank to trace notes and detect suspicious activities, *e.g.*, 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 instance CP, any two honest clients idc_1, idc_2, any corrupted seller ids, we have that:*

$$\begin{aligned} &CP_I[|_{0\le i\le m_1} C\sigma_{idc_1}\sigma_{c_1^i}\sigma_{ids}|_{0\le i\le m_2} C\sigma_{idc_2}\sigma_{c_2^i}\sigma_{ids}| \\ &\qquad C\sigma_{idc_1}\sigma_{c_1}\sigma_{ids}|C_w\sigma_{idc_2}\sigma_{c_2}|S^c\sigma_{ids}|B^c] \\ &\qquad\qquad \approx_l \\ &CP_I[|_{0\le i\le m_1} C\sigma_{idc_1}\sigma_{c_1^i}\sigma_{ids}|_{0\le i\le m_2} C\sigma_{idc_2}\sigma_{c_2^i}\sigma_{ids}| \\ &\qquad C_w\sigma_{idc_1}\sigma_{c_1}|C\sigma_{idc_2}\sigma_{c_2}\sigma_{ids}|S^c\sigma_{ids}|B^c] \end{aligned}$$

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 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 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 4 (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 Studies

In this section, we describe and analyze the online [Cha83] and the offline [CFN90] variants of the protocol, the online protocol implemented by DigiCash [Sch97], and the protocol by Peterson and Poupard [PP97]. To perform the automatic protocol verification we use Proverif [Bla01]. ProVerif uses a process description based on the applied π-calculus, but has syntactical extensions, for example its language is enriched by *events* to check reachability and correspondence properties; besides it can check *observational equivalence* properties. 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 Online Protocol

The Chaum Online 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 "online" 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 $b = \texttt{blind}(x, r)$, 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 $bs = \texttt{sign}(b, skB)$ 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 $s = \texttt{sign}(x, skB)$ on x. The coin consists of the pair $(x, \texttt{sign}(x, sk_B))$.

Payment (and Deposit) Phases. To spend a coin:

1. The client sends the pair $(x, \texttt{sign}(x, sk_B))$ to the seller.
2. After checking the bank's signature, the seller sends the coin $(x, \texttt{sign}(x, sk_B))$ 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.

$$
\begin{aligned}
getmess(sign(m,k)) &= m \\
checksign(sign(m,k),pk(k)) &= m \\
unblind(blind(m,r),r) &= m \\
unblind(sign(blind(m,r),k),r) &= sign(m,k)
\end{aligned}
$$

Table 1. Equational theory for Chaum Online Protocol.

Modeling in ProVerif. The equational theory depicted in Table 4.1 models the cryptographic primitives used within Chaum on-line 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.

Analysis. The result of the analysis is summarized in Table 4.1. We model *Unforgeability* as an injective correspondence between the events *withdraw* and *spend*, they are placed in their appropriate positions, according to the Definition 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[7] thanks to the SAPIC[8] tool. However, it is difficult to model, using Tamarin, rewriting rules that are not subterm-convergent, which is the case of the equation that supports blind signature primitive. 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 online 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

[7] http://www.infsec.ethz.ch/research/software/tamarin.html
[8] http://sapic.gforge.inria.fr/

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

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

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 Protocol

The online Chaum protocol has been implemented by DigiCash[9]. Latter, the specifications of the DigiCash protocol outlined in [Sch97]. It has the same withdrawal phase as Chaum online 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 4.1). The payment and deposit phases are different from those of Chaum online protocol. They are summarized as follows.

Payment (and Deposit) Phases in DigiCash.

1. The client sends the seller $pay = enc((id_s, \mathtt{h}(\textit{pay-spec}), x, \mathtt{sign}(x, sk_B)), pk_B)$ which is the encryption, using the public key of the bank pk_B, of the seller's identity id_s, hash of the payment specification *pay-spec* (specification of the sold object, price etc), and the coin $(x, \mathtt{sign}(x, sk_B))$.
2. The seller signs $(\mathtt{h}(\textit{pay-spec}), pay)$ 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 $\mathtt{h}(\textit{pay-spec})$ 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 online protocol (Table 4.1), the equational theory of DigiCash protocol includes well-known model of the public key encryption represented by the following equation: $dec(enc(m, pk(k)), k) = m$.

[9] DigiCash declared bankruptcy in 1998, and was sold to Blucora.

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

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

Analysis. The result of analysis of DigiCash protocol using ProVerif is summarized in Table 4.2. ProVerif shows the same results as obtained for Chaum online 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 Offline Protocol

The offline 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 $H = (\mathtt{h}(a, c), \mathtt{h}(a \oplus u, d))$, where u is the client identity and $\mathtt{h}$ is a hash function. The client then proceed as in the Chaum online 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 online protocol is that the coin has to be of the following form

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

where the client identity is masked inside it. This aims to reveal the identity if the client 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 $S = \mathtt{sign}(H, sk_B)$. The client also has to keep the random values a, c, d which are used later to spend the coin.

Payment Phase. For a client to spend a coin to a seller:

1. 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 that the seller has to present to the bank at deposit.

Note that, 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. To deposite a coin at the bank:

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 offline 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 model the Chaum off-line protocol in ProVerif, in addition to the equational theory used for the Chaum online protocol (Table 4.1), 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 offline 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 $tr_2 = (h, hx, sign((h, hx), skB), 1, xor(a, u), d)$. Then, 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*. The test emits *event KO* otherwise. To say that *Double Spending Identification* is satisfied, (i) the test $\mathtt{T_{DSI}}$ should 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; (ii) the test should not emit the *event OK* for any two transactions that do not satisfy these conditions. 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

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

Table 4. Analysis of Chaum offline protocol. A (✓) indicates that the property holds. A (×) indicates that it fails. (∗) Only coins with the appropriate form are considered. (†) After applying the countermeasure.

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 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 offline 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. 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. Yet, corrupted sellers can blame a corrupted client who double spends a coin for further spends.

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.

4.4 Protocol by Peterson and Poupard

In this section, we give the description of the protocol due to Peterson and Poupard [PP97], which we call concisely P&P protocol. Then, we present the results of our analysis. The protocol has two variants for electronic purse payments and internet payments, which are slightly differ in few steps. We analyze the internet payments schemes.

The protocol aims, to met the usual security properties, and to prevent extortion attacks, in which a criminal forces the bank to issue coins or reveal secret keys through kidnapping. To this end, the protocol consider an additional authority, the trustee, at which a client has to register using a pseudonymous keypair (PS_x, PS_y). Then, public part of the client's pseudonym PS_y is embedded into the coin. The main rule of the trustee is to take some actions (*e.g.*, publishes list of illegal coins) when an extortion attack is reported. Accordingly, the protocol has an additional phase, *Registration Phase*, in which the client open an account at the bank and registers at the trustee. Note that similar to Chaum protocols, P&P protocol uses a blind signature scheme in order to withdraw anonymous coins, and uses a "challenge-and-response" procedure at payment. In the following, we provide the description of the P&P protocol.

Registration Phase. To open an account:

1. The client id_C registers at the bank and obtains an account number acc_C;
2. The bank stores (id_C, acc_C) in his database.

For the client to identify himself to the trustee:

1. The client and the trustee obtain an authentic session key k_{CT} using an authentic key exchange protocol.
2. The client generates a pseudonymous keypair (PS_x, PS_y), computes the signature $\sigma_C = sign((id_C, PS_y), sk_C)$, and sends $enc((id_C, PS_y, \sigma_C), k_{CT})$ to the trustee, where *enc* is a symmetric encryption function.
3. The trustee verifies the signature σ_c, calculates $\sigma_T = sign(PS_y, sk_C)$, and sends $enc(\sigma_T, k_{CT})$ to the client. The trustee also stores (id_C, PS_y, σ_C) in his database.
4. The client verifies the signature σ_T, and stores all the values.

These steps might be processed several times to obtain several pseudonymous keypairs (PS_x, PS_y).

Withdrawal Phase. The client can withdraw a coin from the bank according to the following steps:

1. The client and the bank first obtain an authentic session key k_{CB} using an authentic key exchange protocol.
2. The client generates random coin c, computes $bc = blind(h(c, PS_y))$ and $e_T = aenc(h(c, PS_y), pk_T)$ where h is a hash function and $aenc$ is a probabilistic asymmetric encryption function, and sends bc and e_T to the bank. Note that, e_T is mainly used in case of an extortion attack against the bank which then sends all the stored values e_T to the trustee.
3. The bank signs the blinded coin by computing $\tilde{\sigma}_B = sign(bc, sk_B)$ and sends it to the client. The bank also subtracts the value of the coin from the client's account acc_C and stores (id_C, bc, e_T).
4. The client unblind the signature $\tilde{\sigma}_B$ to obtain σ_B and verifies it. The client keeps (c, σ_B) as his coin and notices the relation to the tuple (PS_x, σ_T).

Payment Phase. To spend a coin, the client communicates with the seller according to the following steps:

1. The seller sends the client a uniquely generated message $mess$.
2. The client generates the signature $\sigma_{coin} = sign((c, id_S, mess, PS_y), PS_x)$, and sends the payment transcript $(c, PS_y, \sigma_T, \sigma_B, aenc(\sigma_{coin}, pk_S))$.
3. The seller verifies σ_T, σ_B, and σ_{coin}, and stores the payment transcript together with $mess$ in his database.

Note that in addition the protocol consider three extortion cases: extortion of client's secret key PS_x, extortion of bank's secret key sk_B, and extortion of trustee's secret key sk_T. In case of extortion of PS_x, the trustee distributes the corresponding key PS_y among the sellers so that the seller rejects all the coins withdrawn under PS_y. The trustee also issues the signature $sign((id_C, PS_y), sk_T)$ which allows the client exchange unspent coins withdrawn under PS_y at the bank. In case of extortion of bank's secret key, he sends e_T to the trustee which in turn decrypts it and publishes the legal coins among the sellers so that they can verify the coin at payment. In case of extortion trustee's secret key , he signs PS_y and publishes it, then the seller verifies the signature and check if PS_y is among the published list before accepting. We consider these additional checks in our model.

Deposit Phase. To deposite a coin at the bank:

1. The seller sends the tuple $(c, PS_y, id_S, acc_S, mess, \sigma_B, \sigma_T, \sigma_{coin})$ to the bank.
2. The bank verifies the signatures σ_B, σ_T and checks whether the coin was already deposited under the same PS_y. If it is the case, then:
   - If $\sigma'_{coin} = \sigma_{coin}$, the seller is accused for double deposite. The bank sends the tuple (c, σ'_{coin}) to the seller as a proof.
   - If $\sigma'_{coin} \neq \sigma_{coin}$, the bank verifies σ_{coin}. If it is valid, then the coin has been overspent. In this case, the bank sends the trustee the transcripts

$getmess(sign(m,k)) = m$
$checksign(sign(m,k), pk(k)) = m$
$dec(enc(m,k),k) = m$
$adec(aenc(m, pk(k), r), k) = m$
$unblind(blind(m,r),r) = m$
$unblind(sign(blind(m,r),k),r) = sign(m,k)$

Table 5. Equational theory for P&P Protocol.

$(c, PS_y, mess_1, \sigma_B, \sigma_T, \sigma_{coin,1}) \ldots (c, PS_y, mess_k, \sigma_B, \sigma_T, \sigma_{coin,k})$.
To prove that the coin c was overspent, the trustee verifies all transcripts. If they are correct, he looks for the tuple (id_C, PS_y, σ_C) in his database and sends (id_C, σ_C) to the bank. The bank checks the signature σ_C.
Otherwise, If every thing is okay the bank stores the payment transcript in his database and credits the seller's account acc_S by the value of c.

Modeling in ProVerif. To model the P&P protocol in ProVerif, we use the equational theory depicted in Table 4.4. It includes the well-known equational theory for digital signature (functions *sign*, *getmess*, and *checksign*), symmetric encryption (functions *enc* and *dec*), and asymmetric encryption (functions *aenc* and *adec*). We also use two equations for the blinding function (*blind* and *unblind*) similar to the one we use for previous protocols.

Analysis. The result of the analysis is summarized in Table 4.4. Similarly to the previous protocols, we model *Unforgeability* as an injective correspondence between the two events *withdraw* and *spend*. We consider a honest bank and honest seller but corrupted clients. As expected for offline protocols, ProVerif finds an attack against *Unforgeability*. A corrupted client can double spend a coin, However, a collusion between the client and the attacker cannot lead to forging a coin as the correspondence holds without injectivity.

In case of double spending, the bank may receive two transactions of the form $(c, PS_y, mess_1, \sigma_B, \sigma_T, \sigma_{coin,1})$ and $(c, PS_y, mess_2, \sigma_B, \sigma_T, \sigma_{coin,2})$, and additionally he receives from trustee (id_C, σ_C). Then bank checks that the two transactions are valid and that the signature σ_C is computed by the client on (id_C, PS_y). Thus linking PS_y (under which the coin is withdrawn) to the client identity id_C. To verify *Double Spending Identification*, similar to the Chaum offline protocol, we model the test outputs *event OK* when it finds an identity and an evidence, and *event KO* when it fails. ProVerif shows that P&P satisfies *Double Spending Identification*. ProVerif also shows that *Exculpability* is satisfied as the bank cannot forge the client signature σ_C.

For privacy properties, we assume a corrupted bank and a corrupted seller, but honest clients. ProVerif confirms that *Weak Anonymity* is satisfied by P&P protocol. However, *Strong Anonymity* is satisfied only if the client use a distinct

Property	Result	Time
Unforgeability	×	$< 1s$
Double Spending Identification	✓	$< 1s$
Exculpability	✓	$< 1s$
Weak Anonymity	✓	$< 1s$
Strong Anonymity	×	$< 1s$
Strong Anonymity*	✓	$< 2s$

Table 6. Analysis of P&P protocol. A (✓) indicates that the property holds. A (×) indicates that it fails. (∗) Only one coin was withdrawn per a pseudonymous keypair.

pseudonymous keypair for each coin. Otherwise, the seller can easily links two payments to the same client since PS_y is revealed to him at payment.

5 Conclusion

E-cash protocols aim at emulating real cash by offering anonymous payments. Several protocols have been proposed to ensure client's anonymity, as well as, the standard forgery-related properties. However, multiple flaws were discovered on claimed secure protocols. To avoid further bad surprises, formal verification can be used to improve confidence in e-cash protocols.

In this paper, we proposed a formal framework to automatically verify e-cash protocols with respect to multiple essential privacy and forgery-related properties. Our framework relies on the applied π-calculus and uses ProVerif as the verification tool. As a case study, we analyzed the online protocol proposed by Chaum, as well as a real implementation based on it (the DigiCash protocol). We also analyze the offline variant of this system, and the protocol due to Peterson and Poupard. For Chaum online protocol and DigiCash protocol, we found a weakness concerning *Unforgeability*: the attacker can double spend a valid coin when there is a lack of synchronization. Concerning Chaum offline protocol, we re-discover some known attacks which confirms the correctness of our model. Namely we re-discover, the attack against *Unforgeability* (double spending), the attack against *Double Spending Identification* when the coin does not have the proper form, and the attack against *Exculpability* when the counter-measure is not considered. With respect to P&P protocol, we found an expected attack on *Unforgeability* as double spending is usually possible in case of offline protocols. We also found an attack on *Strong Anonymity* when more than one coin is withdrawn under the same pseudonymous keypair.

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, which is based on Tamarin, in order to see how it can help to analyze e-cash protocols.

References

[AA14] Sattar J. Aboud and Ammar Agoun. Article: Analysis of a known offline e-coin system. *International Journal of Computer Applications*, 98(15):27–30, July 2014.

[ABB+05] Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *Computer Aided Verification, 17th International Conference, CAV, UK*, pages 281–285, 2005.

[AF96] 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.

[AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *The 28th Symposium on Principles of Programming Languages, ACM, UK*, pages 104–115, 2001.

[Bla01] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14), Canada*, pages 82–96, 2001.

[Bra94] 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.

[CFN90] 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.

[CG08] Sébastien Canard and Aline Gouget. Anonymity in transferable e-cash. In *Applied Cryptography and Network Security, ACNS, USA*, pages 207–223, 2008.

[Cha83] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of CRYPTO '82*, pages 199–203. Springer US, 1983.

[Cre94] 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.

[CYS05] 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 2005), 28-30 March 2005, Taipei, Taiwan*, pages 131–134, 2005.

[Dam90] 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.

[DC94] 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, Perugia, Italy, May 9-12, 1994, Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 1994.

[DKL15] Jannik Dreier, Ali Kassem, and Pascal Lafourcade. Formal analysis of e-cash protocols. In *SECRYPT 2015 - Proceedings of the 12th International*

Conference on Security and Cryptography, Colmar, Alsace, France, 20-22 July, 2015., pages 65–75, 2015.

[DKR09] 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.

[DY83] D. Dolev and Andrew C. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983.

[Fer94] 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.

[FHY13] 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*, 58(1-2):227–237, 2013.

[KO02] S. Kim and H. Oh. A new electronic check system with reusable refunds. *Int. J. Inf. Sec.*, 1(3):175–188, 2002.

[KT11] 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*, 46(3):325–352, 2011.

[LCPD07] 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 2007, China*, pages 87–103, 2007.

[OS14] Marek R. Ogiela and Piotr Sulkowski. Improved cryptographic protocol for digital coin exchange. In *Soft Computing and Intelligent Systems (SCIS)*, pages 1148–1151, 2014.

[PP97] Holger Peterson and Guillaume Poupard. Efficient scalable fair cash with off-line extortion prevention. In *Proceedings of the First International Conference on Information and Communication Security*, ICICS '97, pages 463–477, London, UK, UK, 1997. Springer-Verlag.

[PSW95] 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.

[PW91] 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.

[Sch97] 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.

[SK14] 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*, ICAET '2014, Singapore, 2014.