High-Level Programming for E-Cash

Pedro Adão[1] ⋆, Cédric Fournet[3,2], and Nataliya Guts[2]

[1] SQIG–Instituto de Telecomunicações and IST, TULisbon, Portugal
[2] MSR-INRIA Joint Centre
[3] Microsoft Research

Abstract. We consider symbolic characterizations of the Compact E-Cash protocol of Camenisch, Hohenberger, and Lysyanskaya [CHL05]. E-cash protocols [Cha82,CFN88] aim at providing robust abstractions for anonymous payment protocols. Properties of interest include, for instance, that users can spend coins anonymously, that users cannot forge coins, and that user should not spend the same coin twice without being eventually caught. These protocols involve sophisticated cryptographic constructions.

Relying on recent work on optimistic value commitment [FGN08], we design a calculus with E-cash primitives. Our calculus has a simple, symbolic semantics; it can be used for programming with E-cash and for reasoning on its properties, while shielding the programmer from its cryptographic implementation.

We consider two variants of the symbolic semantics. An abstract semantics rules out any double spending (by design). A more realistic, intermediate semantics accounts for the possibility of double spending, with reliable detection. We first relate these two semantics, then relate the intermediate semantics to the computational properties of the underlying E-cash protocol.

1 E-Cash Protocols

E-cash is intended for online payments of goods just as ordinary money. But while physical coins are hard to reproduce, electronic representation of a coin cannot be protected from copying. Ideally, we would like e-cash to have all the properties of real money. In particular, users of e-cash should have the anonymity of their spendings protected, and the banks should have a guarantee of double-spending reliable detection.

The basic model of an e-cash protocol includes three categories of principals: banks, users and merchants; merchants may be seen as a subset of users and banks hold the accounts of users and merchants (assumption). A user can *withdraw* an electronic coin from its bank account and *spend* it with a merchant. The merchant can then *deposit* the received coin to its own bank account.

Several implementations of e-cash exist [Cha82,CFN88,OO89,CHL05]. We are interested in abstracting the primitives of the e-cash cryptographic implementations for programming real-world e-cash applications, that is, more than modelling the behaviour of the cryptographic primitives involved in the e-cash protocol, we are interested in the interactions of the generated cryptographic terms in larger applications that provide an e-cash service. More concretely, we are interested in the usability of cryptographic definitions from [CHL05]. This implementation of off-line cash is proved to provide anonymity and exculpability of clients, and guarantees balance and detection of double spenders.

⋆ Partially supported by FCT grant SFRH/BD/8148/2002. Additional support from FEDER/FCT project QSec PTDC/EIA/67661/2006, and FEDER/FCT project KLog PTDC/MAT/68723/2006.

2 Our Technique

We introduce a high-level language, based on a fragment of the applied pi-calculus, that captures the security notions of the referred protocol and we show that runs of the low-level cryptographic protocols can be captured, with overwhelming probability, by the abstract language. We do this in two steps.

The first step is to design a high-level language that guarantees the desirable properties. We consider an abstract semantics that rules out by construction any dishonest behaviour and a more realistic intermediate semantics that accounts for the possibility of double spending, but has reliable detection. We compile the abstract language to the permissive language, and show that transitions of the two match unless a principal did not follow the expected protocol, which can be logged using techniques similar to the ones in [CHL05]. The second step is to encode the intermediate language into a cryptographic setting and to show that transitions of the latter correspond with overwhelming probability to transitions of the former.

The corollary of these two steps is the computational soundness of an abstract language, correct by construction, that captures the computational security properties ensured by the CHL Compact E-cash protocol: Correctness, Balance, Identification of Double Spenders, Anonymity and Exculpability.

3 High-Level Model

3.1 Idealized Model

Our first e-cash representation guarantees balance and anonymity for programs written in it, and models the expected behaviour from honest principals. We consider a variant of applied pi calculus extended with several e-cash primitives. The usual constructs include parallel composition, name restriction, sending and receiving on a channel, conditional process, and replication of a guarded process. Besides, we forbid nondeterministic choice and channel mobility.

E-cash withdraw, spend and deposit protocols are represented with corresponding pairs of input/output namesake primitives. Capabilities of spending or depositing a coin are modeled as consumable communicating processes that can only be put into linear contexts to avoid their duplicating.

P ::=

$\mathsf{withdraw!}\ p\ Cx$	withdraw a coin from bank p
$\mathsf{withdrawn!}\ u\ p$	withdrawl number u
$\mathsf{end?}\ u\ p\ \mathcal{L}$	end of withdrawal u
$\mathsf{withdraw?}\ u\ Cx$	create a coin for user u
$\mathsf{end!}\ u\ p$	end of newloc u
$\mathsf{spend!}\ uMu'$	spend the coin u to pay u' through bank M
$\mathsf{spent!}\ uMu'$	spent coin u with u'
$\mathsf{spend?}\ B\ (\ x\ )\ Cx$	wait a payment through bank B
$\mathbf{expired}\ u$	
$\mathsf{deposit!}\ uMu'$	deposit a spent coin u to bank M (user u' is hidden)
$\mathsf{deposited!}\ uMu'$	deposited coin u to bank M (user u' is hidden)

| deposit? $(x, y)\ P_1\ P_2$ bind a coin, and its depositor in P_1 if it is honest

Semantics We use a reduction semantics to represent the exchange of messages between defined principals. This semantics enforces that coins are only spent once due to the use of linear contexts that only allows the use of a received term once. Asyncrhronicity is needed for privacy, and also by specification, and it is modelled by the use of two rules for input being the first the input to the local buffer and the second consuming the local message.

We use a labelled semantics to reflect communication with an adversary. Input and output labels reflect the knowledge of the adversary. In order to disallow double-spending, we introduce the process outside $b\ B\ U\ S$ that is created when a coin issued by an honest bank is first sent to the environment, and is consumed the first time it is spent.

Equivalence of high level processes Two well-formed processes are equivalent if they have the same sets of reachable labels.

3.2 Intermediate Model

Our second ecash model is more realistic. It shares its grammar with the high level language but allows multiple spending and its detection. This language models cheating in real world ecash protocols, inspired from [CHL05], but is not bound to any particular implementation.

Semantics The reduction semantics is equal to the idealized model except that we do not demand the use of linear contexts, hence allow multiple spending of a withdrawn coin. We keep track of deposited coins so we are able to trace double spending when depositing twice a single coin. As for the labelled semantics, all labelled transitions are the same as in the idealized model except the deposits and spends. We continue to record correctly issued coins with the outside primitive but unlike in the previous semantics, we do not consume outside when spending nor depositing.

3.3 High-Level Properties

With this language we can define some high-level security properties. We present some examples:

- *Correctness.* An honest user can withdraw a coin from an honest bank, according to Withdraw.
  Only the user specified by the bank can receive the spend continuation.
  Only the merchant specified by the client can receive the deposit continuation.
  Only the bank issuer of the coin can accept the deposit of the coin by the merchant.
- *Balance.* For any series of reductions of an initial high level configuration, rule Deposit cannot be applied more often than Withdraw
- *Anonymity of users.* A bank cannot learn anything about a user's spendings

$$p_1@\mathsf{spend!}\ b\ S\ B \mid p_2@0 \approx p_1@0 \mid p_2@\mathsf{spend!}\ b\ S\ B \quad \forall p_1, p_2$$

- *Identification of double-spenders.* In the intermediate semantics, all the deposited coins are recorded, and given two records of a double spent coin, the identity of the user can be extracted using accuse.
- *Weak Exculpability.* Only double-spenders can be accused (by correction of the rule accuse).
- *Strong Exculpability.* an user can only be responsible for coins that he indeed double-spent (each $\mathbf{bad}(b\,,\ U)$ binds user U to a particular coin b).

3.4 Results

- *Relative correctness* If two high level processes are equivalent then they are also equivalent in the weak intermediate semantics, if $A \sim A'$ then $A \sim_w A'$.
- *Completeness* If two well-formed high level processes are equivalent in the intermediate level semantics then they are also equivalent in the high level semantics, if $A \sim_i A'$ then $A \sim A'$.
- *Progress* If P is cheated at bank B on coin b, and if B is honest, then spender of the coin will be accused by B.
- *Traces* For any source system, if a trace is possible in the high level semantics, then it is still possible in the intermediate semantics.

4 Low-Level Model

We implement our intermediate processes P as a PPT Turing machine and concrete cryptography. We base our implementation on the specification of E-Cash primitives given by Camenisch et al. [CHL05]. Each machine can perform any of the three roles of the protocol and is composed of the running process p and three "cryptographic boxes", called $\mathsf{User}, \mathsf{Merchant}$ and Bank that run each of the roles. Each of these boxes has access to a *Cache* that keeps track of the runtime state of these protocols, that is, for any instance of the protocol it contains all messages that were exchanged previously.

Communications Model The user role, as well as the merchant role, of each machine shares an authenticated channel with each bank role of each other machine to be used in the withdrawal and deposit protocols respectively. Each merchant role has a unique channel for incoming payments that, to achieve anonymity, is connected to a multiplexer that is then connected by a channel to each user role. All users are connected among them. Communications with the adversary are done through another multiplexer that shuffles messages in order to prevent traffic analysis.

Execution Model Systems consist of a finite number of communicating principals that are complying with the high level semantics (they may double spend). When activated, a machine reads one message from its input tape and processes it: if it is an e-cash message, routes it to the appropriate crypto box; if it is a communication message, sends it to the running process. All machines should run to completion, ie, consume all messages in its input tape and write all output messages in the input tapes of the intended receivers. Whenever an e-cash primitive is in evaluation position of the running process, the process either sends or waits for a signal from the corresponding crypto box. Whenever all machines have completed, all messages to principals that are not defined in the system are shuffled, and given to the adversary. The adversary controls some compromised principals but not the network.

4.1 Results

- *Soundness* If two intermediate processes are equivalent then with an overwhelming probability their low level implementations are also equivalent.
- *Lifting of traces* For any source system A, shadow D, making transitions $\mathsf{M}(A, \mathsf{D}) \overset{\sigma_1 \sigma_2}{\rightsquigarrow} \mathsf{M}'$, there exist a source system A' and shadow D' such that $A \xrightarrow{\phi} A'$ and $\mathsf{M}' = \mathsf{M}(A', \mathsf{D}')$ where D' is derived from $\mathsf{D}, \sigma_1, \sigma_2$ and $\ulcorner \phi \urcorner = \sigma_2$.
- *Completeness* If the implementations of two intermediate processes are equivalent then with overwhelming probability those processes are also equivalent.

5 Summary

We define a 3-layer cake to reason about E-Cash protocols that allow us to symbolically reason about e-cash services based on e-cash primitives rather than a specific e-cash protocol. We reason about cryptographic actions rather than cryptographic terms. To achieve this we consider a symbolic layer where bad behaviours do not occur by construction, and introduce an intermediate symbolic layer where probabilities are discarded. We show that all intermediate runs are "well-behaved" or a dishonest behaviour may be detected. After that, we show that the low-level crypto layer is correctly abstracted by the "well-behaved" semantics.

Finally, we expect that this same technique may be applied to other similar cryptographic tasks such as e-voting.

References

[AF06] Pedro Adão and Cédric Fournet. Cryptographically sound implementations for communicating processes. In *ICALP (2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 83–94. Springer, 2006.

[CFN88] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *CRYPTO*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327. Springer, 1988.

[Cha82] David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1982.

[CHL05] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 302–321. Springer, 2005.

[FGN08] Cédric Fournet, Nataliya Guts, and Francesco Zappa Nardelli. A formal implementation of value commitment. In Sophia Drossopoulou, editor, *Programming Languages and Systems (ESOP'08)*, volume 4960 of *LNCS*, pages 383–397. Springer, 2008.

[OO89] Tatsuaki Okamoto and Kazuo Ohta. Disposable zero-knowledge authentications and their applications to untraceable electronic cash. In *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 481–496. Springer, 1989.