On Tamper-Resistance from a Theoretical Viewpoint - The Power of Seals

Paulo Mateus
SQIG /Instituto de Telecomunicações - IST/TULisbon

Serge Vaudenay
EPFL
CH-1015 Lausanne, Switzerland

2009

Abstract

Tamper-proof devices are pretty powerful. They can be used to have better security in applications. In this work we observe that they can also be maliciously used in order to defeat some common privacy protection mechanisms. We propose the theoretical model of *trusted agent* to formalize the notion of programmable secure hardware. We show that protocols not using tamper-proof devices are not deniable if malicious verifiers can use trusted agents. In a strong key registration model, deniability can be restored, but only at the price of using key escrow. As an application, we show how to break invisibility in undeniable signatures, how to sell votes in voting schemes, how to break anonymity in group/ring signatures, and how to carry on the Mafia fraud in non-transferable protocols. We conclude by observing that the ability to put boundaries in computing devices prevents from providing full control on how private information spreads: the concept of sealing a device is in some sense incompatible with privacy.

1 Introduction

Tamper-proof hardware devices have been used quite massively in industrial and commercial applications. There exists a wide spectrum of tamper-proof devices, ranging in their price and security, from simple smartcards to the IBM 4758, which has several physical penetration sensors, including temperature, radiation, pressure, etc. Clearly, people are currently surrounded by devices (aimed at) instantiating trusted agents. People wear smart cards, secure tokens, their PCs have Trusted Computing Platforms, their media readers have secure hardware to deal with DRMs, their iPhones have a self-blocking secure hardware, passports have secure RFID tags, etc.

So far, secure hardware devices have been used to implement some strong security protocols with the hypothesis that they are tamper-resistant. The idea of using tamper-proof devices to realize cryptographic functionalities goes back (at least) to 1986 [10]. Due to existence of all side channel attacks, whether tamper resistance is possible in practice is still an open question. Current allegedly tamper-resistant devices are (at

least) trusted by banks, mobile telephone operators, companies selling access control devices, software companies, media content providers, hardware manufacturers, governments, and so on. It is unlikely that none of these organizations would ever try to take any malicious advantage out from their devices. So, assuming some adversaries would use tamper-proof devices for attacks is a legitimate assumption. In this paper we show how to make several privacy attacks using trusted tamper-proof devices.

In this work, we formalize the notion of programmable secure hardware by introducing the *trusted agent model*. Informally speaking, the trusted agent model consists in assuming that it is possible to acquire a trusted device (agent) that runs honestly a known program in a secure environment (tamper proof) without any way of running another program. At the first time it is switched on, we can load a code whose digest will be permanently displayed. Later, we can interact with the device through the interface defined by this program only. Quite importantly, every output will be appended to the digest of the original program so that someone looking at the display is ensured that the output is produced by a device having been set up with a program of displayed digest. We show that within this model, it is possible

- to transfer proofs of zero-knowledge protocols after completion (in particular: to transfer the verification of an invisible signature);
- to register rogue public keys and prove the ignorance of a secret key (which then can be used to break anonymity in ring signatures or non-transferability mechanisms);
- to sell ballots in e-voting systems.

In a nutshell, for any interactive proof protocol, we can load the verifier algorithm in a trusted agent and make a malicious verifier relay protocol messages between the prover and the trusted agent. Afterward completion, the agent ends up in a state which testifies that the proof protocol was correctly run and provide some kind of forensic evidence. Clearly, such a honest device could be used to defeat the invisible signature paradigm [6] when maliciously used. One could say that this trivial attack could be defeated by classical non-transferability techniques like having a Public Key Infrastructure (PKI) for verifiers [2, 11, 15, 16]. However, this would work only if the prover is convinced that the verifier possesses himself a secret key. A trusted agent could still be maliciously used to register a key whose secret part would be ignored by the verifier. Later, the agent could prove that the verifier must ignore the secret key and continue to defeat non-transferability. Finally, the only key registration model which could fix this would imply some kind of key escrow: some information making the registrating authority able to impersonate the verifier would eventually have to leak in order to thwart the previous attack. Key escrow however leads us to other privacy concerns.

Another possible use of registering a public key whose secret component is sealed in a trusted tamper-proof hardware would be to break anonymity in group signatures or ring signatures [23]. Interestingly, it makes it possible to *prove ignorance*. It could also be used in voting systems and open the door to vote selling.

While it is debatable if the trusted agent model is realizable or not, assuming it cannot be used by adversaries is a much greater error than assuming that it can. For

this reason, we believe that cryptographers should mind the proposed trusted agent model when designing future protocols.

Related work. Classical ZK proof systems fulfill a privacy property called *deniability* [21] stating that the verifier cannot prove knowledge to a third party after interacting with the prover. That is, the verifier cannot transfer the proof upon completion. The more general concept of *non-transferability* is also central in some cryptographic schemes, such as invisible signatures [6][1] that use interactive verification in order to prevent the signature to be authenticated to an unauthorized third party. A different way to enforce deniability of a signature is to use a group or ring signature [23] between the signing party and the verifier. In this case, the signer can deny the signature by claiming that it was computed by the other party.

Several flaws have been found to non-transferability protocols, and improvements have been proposed (see e.g. [2, 11, 15, 16]). The attacks are focused in adversaries that are online with the verifier during the interaction period. For offline attacks, it is accepted that the protocols are secure. However, herein we will present an offline attack that will render several non-transferability protocols useless under the assumption that the participants can trust tamper-proof devices.

The idea of using tamper-proof hardware to transfer proofs of ZK protocols was first introduced in the context of quantum memory [18, 19].

In general, setup phases in cryptographic protocols is a critical issue. Participants are often assumed to securely register their public keys, although doing so is not trivial. Key setup is a problem for the Universal Composability (UC) framework by Canetti [3]. For instance, the key registration model by Barak, Canetti, Nielsen and Pass [1] assumes that the secret key of honest participants is safely stored by the key registration authority. In [22], Ristenpart and Yilek considered several variants of key registration protocols and have shown tricky interference with the security in several group signature protocols. They noticed that security proofs often assume that all participants send their secret keys to a trusted authority in a KOSK model (as for *Knowledge Of Secret Key*) although some signature schemes could still be secure in a less demanding key registration process such as producing a self-signed certificate for the public key, what they call the POP (as for *Proof Of Possession*). Our results show that POP is either not enough in the trusted agent model, or compromises some other cryptographic property.

Katz [17] used another approach consisting in assuming the existence of tamper-proof hardware tokens. These tokens could be used to achieve commitment, thus any well-formed functionality. Contrarily to these hardware tokens, we assume that trusted agents are private (namely: their holders do not give them to another user) and display the initial code (or its digest) so that any other party can trust that it is in a state which is a consequence of having set it up with this code. The question whether a tamper-proof hardware can be trusted to run what it is supposed to is discussed e.g. in [14].

In [21], Pass introduced the notion of deniable zero-knowledge which is immune to offline proof transfer. ZK in the standard model is essentially deniable. However,

[1] As suggested by several authors, we use the term of *invisible signature* to designate what is more often called *undeniable signature* since the term *undeniable* is a little confusing, especially when we introduce the notion of deniability.

zero-knowledge in the common reference string (CRS) model is not always deniable.

Structure of the paper. The paper is organized as follows, in Section 2 we introduce the trusted agent model and the nested trusted agent model. In Section 3 we study deniability. We show that deniability is impossible if honest participants do not use trusted agents but the malicious verifier does. We show that deniability is possible when the prover uses trusted agents. In other cases, it is impossible in the nested trusted agent model, and possible in the trusted agent model. Section 4 studies a key registration model. It shows that key registration with key escrow makes non-transferability possible. We provide examples of malicious use of trusted agents in Section 5. Finally, we draw some conclusions in Section 6.

2 The Trusted Agent Model

Multiparty computation model. In a multiparty setting, several participants or functionalities[2] run different algorithms and can communicate using pairwise communication channels. Channels are assumed to be secure in the sense that leakage or corruption in transmitted messages can only be made by one of the two end participants on this channel. We consider a static adversarial model in which participants are either honest or corrupted. Honest participants run predefined algorithms whereas corrupted participants may run arbitrary algorithms and talk to an (imaginary) adversary to collude. We use calligraphic characters (e.g., $\mathcal{P}_V$ or $\mathcal{F}_{\mathsf{TA}}$) to denote participants and functionalities and capital characters (e.g., V or M) to denote the algorithms they run. By convention we will denote with a star $*$ the corrupted participants in a static model. Sometimes, a participant $\mathcal{P}$ invoking a functionality O will be referred to $\mathcal{P}$ querying an *oracle* O and we will write $\mathcal{P}^O$ for this type of communication. Later, a trusted agent will be defined by a functionality and used as an oracle. At the beginning, an arbitrary environment $\mathcal{E}$ sends input to all participants (including the adversary and functionalities) and collect the output at the end.

We stress that we do not necessarily assume that malicious participants have the same privileges as honest participants, which means that they can have access to different sets of functionalities. For instance, a malicious participant may use a trusted agent as a tool for cheating while we would not want a honest one to need an extra device.

Recall that an interactive machine is a next-message deterministic function applied to a current *view*. The view of the algorithm is a list containing all inputs to the machine (including the random coins) and all messages which have been received by the machine (with a reference to the communication channel through which it was delivered so that they can see which ones come from a trusted agent). The view is time dependent and can always be expanded by adding more messages.

We denote by $P \leftrightarrow V$ two interactive algorithms P and V interacting with each other, following a given protocol. When there is a single message sent by e.g. P to V, we say that the protocol is non-interactive and we denote it by $P \rightarrow V$. If O_P (resp. O_V) is the list of functionalities that participant $\mathcal{P}_P$ (resp. $\mathcal{P}_V$) may invoke when running

[2]Following the traditional terminology of universal composability [3], a functionality is a virtual participant performing honestly a specific cryptographic task.

the algorithm P (resp. V) we denote by $P^{O_P} \leftrightarrow V^{O_V}$ the interaction. More precisely, we denote by $P^{O_P(r_{OP})}(x_P; r_P) \leftrightarrow V^{O_V(r_{OV})}(x_V; r_V)$ the experiment of running P with input x_P and random coins r_P with access to O_P initialized with random coins r_{OP} and interacting with V with input x_V and random coins r_V with access to O_V initialized with random coins r_{OV}. We denote by $\mathsf{View}_V(P^{O_P(r_{OP})}(x_P; r_P) \leftrightarrow V^{O_V(r_{OV})}(x_V; r_V))$ the *final view* of V in this experiment, i.e. x_V, r_V and the list of messages from either P or O_V.

The trusted agent model. We assume that it is possible to construct a trusted device (agent) that runs honestly a known program (a minimal boot loader) in a secure environment (tamper proof) without any way of running another program. Moreover we assume that the device's memory is private, and that the only way to interact with the device is by using the interface defined by the program. A device is attached to a participant called its *holder*. He entirely controls the communication with it. The holder may however show the display of the device to another participant which would give him some kind of evidence of the outcome produced by a trusted agent. Below, we model trusted agents in a similar way as Katz's secure tokens [17]. Differences will be discussed in the full version of the paper.

We consider (probabilistic) interactive Turing machines with four kinds of tapes: the input tape, the working tape, the output tape, and the random tape. We define their *state* by the state of the automaton and the content of the working tape. We consider a programming language to specify the *code* of the transition function of the Turing machine and its *initial state*. All trusted agents are modeled by a functionality $\mathcal{F}_{\mathsf{TA}}$. To access to a particular trusted agent, we use a sid value. For each used sid, $\mathcal{F}_{\mathsf{TA}}$ stores a tuple in memory of form $(\mathsf{sid}, \mathcal{P}, r, C, \mathsf{state}, \mathsf{out})$, where $\mathcal{P}$ identifies the holder of the trusted agent, r denotes its random tape, C the loaded code to be displayed, state its current state, and out its output tape. $\mathcal{F}_{\mathsf{TA}}$ treats the following queries.

Query $\mathsf{SEND}(\mathsf{sid}, m)$ **from participant** $\mathcal{P}$**:** If there is a tuple $(\mathsf{sid}, \mathcal{P}', \ldots)$ registered with a participant $\mathcal{P}' \neq \mathcal{P}$, ignore the query. Otherwise:

- If there is a tuple with correct participant, parse it to $(\mathsf{sid}, \mathcal{P}, r, C, \mathsf{state}, \mathsf{out})$ and set in to the value of m.
- If there is no tuple registered, interpret m as a code C. Extract from it the value state of its initial state. Then set in and out to the empty string. Pick a string r of polynomial length at random. Then, store a new tuple $(\mathsf{sid}, \mathcal{P}, r, C, \mathsf{state}, \mathsf{out})$.

Then, define a Turing machine with code C and initial state state, random tape set to r, input tape set to in, and output tape set to out. Then, reset all head positions and run the machine until it stops, and at most a polynomial number of steps. Set state to the new state value, set out to the content of the output tape, and update the stored tuple with the new values of state and out.

Note that the registered $(\mathsf{sid}, \mathcal{P}, r, C)$ are defined by the first query and never changed.

Query $\mathsf{SHOWTO}(\mathsf{sid}, \mathcal{P}')$ **from participant** $\mathcal{P}$**:** If there is no tuple of form $(\mathsf{sid}, \mathcal{P},$

$r, C, \mathsf{state}, \mathsf{out}$) with the correct $(\mathsf{sid}, \mathcal{P})$, ignore. Otherwise, send to $\mathcal{P}'$ the pair formed by the code C and the content out.

Here, the holder $\mathcal{P}$ asks for the creation of a new trusted agent by invoking a fresh instance sid of the functionality which becomes an agent. The holder is the only participant who can send messages to the agent. The holder can define to whom to send response messages by the SHOWTO message. (Incidentally, the holder can ask to see the output message himself.) The response message (as *displayed* on the device) consists of the originally loaded code C and the current output out. Since the channel from $\mathcal{F}_{\mathsf{TA}}$ to $\mathcal{P}'$ is secure, $\mathcal{P}'$ is ensured that some instance of $\mathcal{F}_{\mathsf{TA}}$ (i.e. some trusted agent) was run with code C and produced the result out. Showing a trusted agent may provide a simple way to authenticate some data. By convention, we denote by $[C : \mathsf{out}]$ an incoming message from $\mathcal{F}_{\mathsf{TA}}$ composed by a code C and a value out. The action of *checking* $[C : \mathsf{out}]$ means that the receiver checks that it comes form $\mathcal{F}_{\mathsf{TA}}$, that C matches the expected code, and that out matches the expected pattern from the context.

One important property of this functionality is that it is well-formed. We say that a list of oracles O is *well-formed* if for any pair $(\mathcal{P}_0, \mathcal{P}_1)$ of participants and any algorithm M with access to O, there exists an algorithm S with access to O such that the two following experiments are indistinguishable from the environment:

1. $\mathcal{P}_0$ runs M^O and $\mathcal{P}_1$ runs an algorithm doing nothing.
2. $\mathcal{P}_0$ runs and algorithm defined by
   - for any incoming message m from $\mathcal{P} \neq \mathcal{P}_1$, $\mathcal{P}_0$ sends $[\mathcal{P}, m]$ to $\mathcal{P}_1$;
   - for any incoming message $[\mathcal{P}, m]$ from $\mathcal{P}_1$, $\mathcal{P}_0$ sends m to $\mathcal{P}$;
   - upon message $[\mathsf{out}, m]$ from $\mathcal{P}_1$, the algorithm ends with output out.

   The participant $\mathcal{P}_1$ runs S^O.

Typically, S emulates the algorithm M by treating all messages forwarded by $\mathcal{P}_0$ and by using $\mathcal{P}_0$ as a router. This means that the output of O is not modified if the algorithm M is run by $\mathcal{P}_0$ or by $\mathcal{P}_1$. Informally, being well-formed means that the distribution of roles among the participants does not affect the behavior of O. An example of a functionality for which this is *not* the case is a key registration functionality who registers the name of the sending participant and reports it to a directory. So, the adversary could check if the key was registered by $\mathcal{P}_0$ or by $\mathcal{P}_1$ and tell it to the environment. As for $\mathcal{F}_{\mathsf{TA}}$, it checks that messages come from the same holder but his identity has no influence.

Relevance of the model in practice. Our model for trusted agent could easily be implemented (assuming that tamper-resistance can be achieved) provided that we could trust a manufacturer and that nobody could counterfeit devices. Obviously this is a quite strong assumption but this could make sense in applications where there is a liable entity which must be trusted. For instance, digital payment relies on trusted agents issued by a liable bank: credit cards have a tamper-proof embedded chips and e-banking is often based on trusted secure tokens such as secureID. Nation-wide e-governance could be based on protocols using trusted agents. It is already the case for

passports, ID documents, or health cards with tamper-proof RFID chips. In this paper, we demonstrate that such devices can be used for malicious reasons and not only to protect the user against attacks. We show in Appendix A that our trusted agent model can perform bit commitment. Following Katz's reasoning [17], since we can realize commitments in the $\mathcal{F}_{\mathsf{TA}}$-hybrid model we can also realize many protocols which suffer from being impossible to realize in the bare model. Namely, we can realize any well-formed functionality [5].

Nested trusted agents. Regular trusted agents run Turing machines which cannot interact with other functionalities during their computation time. We can consider more general trusted agents who can use external oracles. Typically, we will consider generalized trusted agents (which we call *nested* trusted agents) who can become the holder of another trusted agents. To take an example, a (human) holder may communicate with a trusted agent "of depth 2" modeled by the functionality $\mathcal{F}_{\mathsf{TA}}^{2}$. The participant may then receive $[C : \mathsf{out}]$ messages from $\mathcal{F}_{\mathsf{TA}}^{2}$ (if the holder asked to) or from the functionality $\mathcal{F}_{\mathsf{TA}}^{1}$ of regular trusted agents (upon the request by the nested trusted agent).

Formally, if O is a list of functionalities, we consider the functionality $\mathcal{F}_{\mathsf{TA}}^{O}$ which looks like $\mathcal{F}_{\mathsf{TA}}$ with the difference that the running code C can now send messages to all functionalities in O. Note that if $O = \perp$ this means that no oracle is used and then, $\mathcal{F}_{\mathsf{TA}}^{O}$ is the regular $\mathcal{F}_{\mathsf{TA}}$ functionality. When O designates a trusted agent functionality, we assume that $\mathcal{F}_{\mathsf{TA}}^{O}$ keeps a record of which instance sid of $\mathcal{F}_{\mathsf{TA}}^{O}$ queries which instance sid' of O so that only the holder device $\mathcal{F}_{\mathsf{TA}}^{O}(\mathsf{sid})$ can communicate to a designated trusted agent $O(\mathsf{sid}')$, just like for human holders. Equivalently, we can say that $\mathcal{F}_{\mathsf{TA}}^{O}$ works by cloning itself in clones $\mathcal{F}_{\mathsf{TA}}^{O}(\mathsf{sid})$.

We define $\mathcal{F}_{\mathsf{TA}}^{0} = \perp$ (this is a dummy functionality doing nothing) and $\mathcal{F}_{\mathsf{TA}}^{n} = \mathcal{F}_{\mathsf{TA}}^{\mathcal{F}_{\mathsf{TA}}^{n-1}}$ iteratively. We have $\mathcal{F}_{\mathsf{TA}}^{1} = \mathcal{F}_{\mathsf{TA}}$. We further define $\mathcal{F}_{\mathsf{NTA}} = \mathcal{F}_{\mathsf{TA}}^{\mathcal{F}_{\mathsf{NTA}}}$. That is, instances of $\mathcal{F}_{\mathsf{NTA}}$ can invoke $\mathcal{F}_{\mathsf{NTA}}$. We obtain a hierarchy of functionalities starting with $\perp$ and $\mathcal{F}_{\mathsf{TA}}$ and ending with $\mathcal{F}_{\mathsf{NTA}}$. To simplify, we consider $\mathcal{F}_{\mathsf{TA}}^{n}$ as a restricted usage of $\mathcal{F}_{\mathsf{NTA}}$ for all n. That is, holders load nested trusted agents with codes which are clearly made for an agent of a given depth. A participant receiving a message $[C : \mathsf{out}]$ from $\mathcal{F}_{\mathsf{NTA}}$ can see that it is from a trusted agent of depth bounded by n.

3 Forensic Attacks Based on a Trusted Witness (Deniability Loss)

We recall here the definitions of a hard predicate and a zero-knowledge argument of knowledge system [12, 13]. We slightly adapt it so that the prover and the verifier can talk to a specific list of oracles (typically: trusted agents). If a list is specified as $\perp$ or unspecified, we consider that no oracle is used. Quite importantly, we do not necessarily assume that honest and malicious verifiers have access to the same oracles.

Definition 3.1 (Hard predicate) *Let $R(x,w)$ be a predicate relative to a* statement *x and a* witness *w. We say that R is a hard predicate if (1) there is a polynomial-time Turing machine A such that $A(x;w)$ yields 1 if and only if $R(x,w)$ and (2) there is no*

probabilistic polynomial-time Turing machine B such that for any x, $B(x;r)$ returns w such that $R(x,w)$ with non-negligible probability (over the random coins r).

Definition 3.2 (Zero-knowledge argument) *Let $R(x,w)$ be a predicate relative to a* statement *x and a* witness *w, O_P, O_V, O_V^* be three lists of oracles initialized using a list of random coins r_I. An* argument of knowledge *for R relative to (O_P, O_V) is a pair (P^{O_P}, V^{O_V}) of polynomial-time interactive machines $P^{O_P}(x,w;r_P)$ and $V^{O_V}(x,z;r_V)$ such that: x is a common input; P has a secret input w; V has an auxiliary input z and produces a binary output (accept or reject); and, moreover, the system fulfills the following properties:*

- Completeness*: for any r_I, r_P, r_V, x, w, z such that $R(x,w)$ holds, the outcome of interaction $P^{O_P}(x,w;r_P) \leftrightarrow V^{O_V}(x,z;r_V)$ makes V accept.*
- Soundness*: there exists a polynomial-time algorithm E (called* extractor*) which is given full black-box access*[3] *to the prover such that for any x and z, any polynomial-time algorithm P^* with access to O_P, if the probability (over all random coins) that $P^{*O_P}(x;r_P) \leftrightarrow V^{O_V}(x,z;r_V)$ makes V accept is non-negligible, then $E^{P^*}(x;r)$ produces w such that $R(x,w)$ with non-negligible probability (over r).*

The argument system is called zero-knowledge *(ZK) relative to O_V^* (or O_V^*-ZK) if for any polynomial-time algorithm $V^{*O_V^*}$ with access to O_V^* there exists a polynomial-time algorithm $S^{O_V^*}$ (called* simulator*), which could be run by the verifier, such that for any x, w, and z such that $R(x,w)$, the experiments of either computing $\mathsf{View}_V(P^{O_P}(x,w;r_P) \leftrightarrow V^{*O_V^*}(x,z;r_V))$ or running $S^{O_V^*}(x,z;r)$ produce two random (over all random coins) outputs with indistinguishable distributions.*

As shown by the following classical lemma, our definition of zero-knowledge is essentially *deniable* because the simulator can be run by the verifier [21]. This means that V^* cannot produce some y which could serve to feed a malicious prover P^*.[4]

Lemma 3.3 *Let (P^{O_P}, V^{O_V}) be an argument of knowledge system for R. The system is O_V^*-ZK if and only if for any polynomial-time algorithm $V^{*O_V^*}$ producing a final output y, there exists a polynomial-time algorithm $S^{O_V^*}$ which could be run by the verifier such that for any x, w and z such that $R(x,w)$, the experiments of either running $P^{O_P}(x,w;r_P) \leftrightarrow V^{*O_V^*}(x,z;r_V)$ and getting the final output y of V^* or running $S^{O_V^*}(x,z;r)$ produce two random outputs with indistinguishable distributions.*

In the next lemma, we show that if the honest verifier has access to O_V, a malicious verifier V^* holding a nested trusted agent N^{O_V} can defeat deniability.

Lemma 3.4 (Transference Lemma) *Let O_P, O_V be any oracle lists. We assume O_V is well-formed. Let N^{O_V} be a nested trusted agent with O_V embedded. Let (P^{O_P}, V^{O_V})*

[3] This means that E can call P^* as a subroutine, choose all inputs including the random tape (i.e. it has "rewindable access"), see all outputs including queries to the oracles invoked by P^* and simulate their responses.

[4] This notion of deniability is sometimes called *self-simulatability* [1] to avoid confusion with other notions of deniability which are used in encryption or signature.

be an argument of knowledge for R. We assume that V only receives messages from either O_V or the prover $\mathcal{P}_P$. There exists a non-interactive argument of knowledge $(Q^{O_P,N^{O_V}},W)$ for R and a malicious verifier $V^{*N^{O_V}}$ producing a final string y such that the random variables

$$\mathsf{View}_W\left(P^{O_P}(x,w;r_P) \leftrightarrow V^{*N^{O_V}}(x;r_V) \rightarrow W(x;r_W)\right) \quad \textit{and}$$

$$\mathsf{View}_W\left(Q^{O_P,N^{O_V}}(x,w;r_P) \rightarrow W(x;r_W)\right)$$

are indistinguishable.

Proof: We define a code C implementing the algorithm V^{O_V} to be run by N^{O_V}. The code terminates the protocol by yielding either "x accepted" or "abort".

To construct $Q^{O_P,N^{O_V}}$, we first load N^{O_V} with the same C. Then, we simulate the protocol between P^{O_P} and N^{O_V}. Finally, Q sends SHOWTO $\mathcal{P}_W$ where $\mathcal{P}_W$ is the participant running W. To define W, we just make it check that the message is $[C : x \text{ accepted}]$ with the correct C and x and that the message comes from a trusted agent. Clearly, $(Q^{O_P,N^{O_V}},W)$ is an argument of knowledge for R: it is complete, and for soundness we observe that if W accepts, then it must have received $[C : x \text{ accepted}]$ from a trusted agent who thus must have run the code C and complete the proof verification. This means that from the malicious $Q^{*O_P,N^{O_V}}$ interacting with N^{O_V} we can first extract an algorithm P^{*O_P} to complete the proof with V^{O_V} and then extract a witness.

To construct V^{*N}, we simply let V^* load N^{O_V} with C and relay messages between P^{O_P} and N^{O_V}. Finally, V^* sends SHOWTO $\mathcal{P}_W$. Clearly, $\mathsf{View}_W(P^{O_P} \leftrightarrow V^{*N} \leftrightarrow W)$ and $\mathsf{View}_W(Q^{O_P,N} \leftrightarrow W)$ are identically distributed. QED

QED

For $O_V = \perp$, this result tells us that we can make any argument of knowledge non-interactive by using a trusted agent. In other words, a malicious verifier V^* equipped with a trusted agent O, after interacting with the prover P, can behave as a prover Q to transfer non-interactively the argument of knowledge to any verifier W offline. This is done by simply certifying a correct execution of V by O. Clearly, trusted agents make the whole notion of NIZK pretty simple to achieve. This further leads us to making deniable zero-knowledge collapse.

Theorem 3.5 *Let O_P, O_V be any oracle lists. We assume O_V is well-formed. Let N^{O_V} be a nested trusted agent with O_V embedded. Let R be any hard predicate. No argument of knowledge (P^{O_P},V^{O_V}) for R such that V only receives messages from O_V or the prover $\mathcal{P}_P$ is N^{O_V}-ZK.*

In particular, if O is a trusted agent, no (P,V) argument for R is O-ZK. In clear, if a malicious verifier can use a trusted agent but the honest participants do not, the argument system is not zero-knowledge.

Proof: Let (P^{O_P},V^{O_V}) be a N^{O_V}-ZK argument of knowledge for a relation R such that V only receives messages from O_V or the prover $\mathcal{P}_P$. We define V^*,Q,W by Lemma 3.4. Due to Lemma 3.3, there must exist a simulator $S^{N^{O_V}}$ making a string y without interacting with P^{O_P}. This string is indistinguishable from the one generated

by V^{*N} after the interaction with P^{O_P}, so it must be accepted by W. Since $(Q^{O_P,N^{O_V}},W)$ is an argument of knowledge for R, we can use an extractor on $S^{N^{O_V}}$ to get a witness w for x. This contradicts that R is hard. QED

QED

Our result shows the inadequacy of deniable zero-knowledge as soon as adversaries can use trusted agents. It does not mean that deniable zero-knowledge is impossible in this model since honest participants could also use trusted agents to protect against transference attacks.

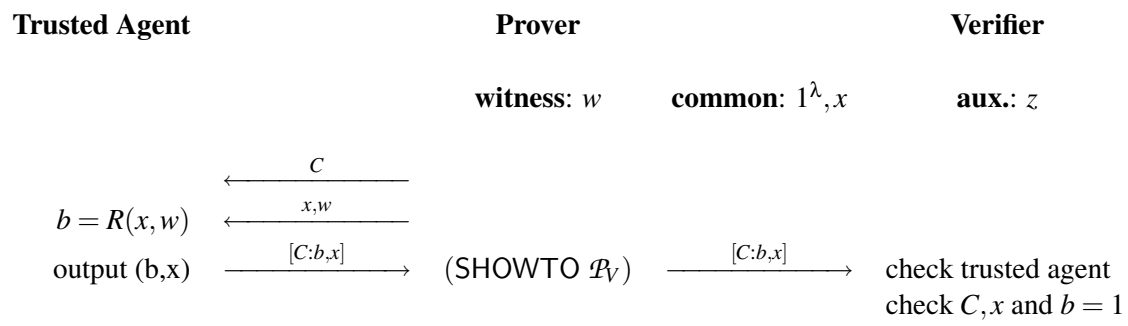

Figure 1: UC-realization of the $\mathcal{F}_{\mathsf{ZK}}$ in the $\mathcal{F}_{\mathsf{TA}}$-hybrid model.

Indeed, if the prover uses a trusted agent which can directly send messages to the verifier (which is excluded in the hypothesis of Theorem 3.5), then it is possible to realize deniable zero-knowledge as depicted in Figure 1.[5] The code C makes $\mathcal{F}_{\mathsf{TA}}$ waits for (x,w), computes $b = R(x,w)$, and outputs (b,x). Clearly, for this protocol to work it is essential that in the last step, the message $[C : b,x]$ reaches V from the prover's TA via an authenticated channel. The protocol is sound, because if the TA yields $[C : 1,x]$, it must be the case that it received w such that $R(x,w) = 1$ and the extractor can see messages from P to it. Moreover, it is deniable, because $\mathsf{View}_V(P^{\mathcal{F}_{\mathsf{TA}}}(x,w;r_P) \leftrightarrow V^*(x,z;r_V)) = \langle x,z,r_V,\mathcal{F}_{\mathsf{TA}} : [C : 1,x]\rangle$ and this string could be forged from x,z,r_V by the verifier running $S(x,z;r_V)$.

If the prover uses no oracle, the situation is more complicated. Actually, in the asymmetric case where the verifier uses a nested trusted agent of depth n but the malicious verifier uses a nested trusted agent of higher depth, no zero-knowledge is feasible due to Th. 3.5. Note that the attack in Th. 3.5 requires an oracle N^{O_V} for V^* with higher depth than O_V for V. In fact, in symmetric cases where both verifiers use a nested TA $\mathcal{F}_{\mathsf{TA}}^n$ with same depth n, zero-knowledge is possible.[6] In symmetric cases where verifiers use $\mathcal{F}_{\mathsf{NTA}}$ (i.e. nested trusted agents of unbounded depth), no zero-knowledge is possible since $\mathcal{F}_{\mathsf{TA}}^{\mathcal{F}_{\mathsf{NTA}}} = \mathcal{F}_{\mathsf{NTA}}$. Feasible ZK results are summarized in the following table.

[5]More formally: we can realize, in the sense of the universal composability framework, a zero-knowledge functionality $\mathcal{F}_{\mathsf{ZK}}$ in the $\mathcal{F}_{\mathsf{TA}}$-hybrid model.

[6]We can show this by having V make a commitment (using a TA) to a random challenge prior to a Σ-protocol so that we transform a honest-verifier zero-knowledge protocol into a full zero-knowledge one.

oracle for P	oracle for V	oracle for V^*	feasibility	comment
none	none	none	yes	classical situation
none	$\mathcal{F}_{\mathsf{TA}}^n$	$\mathcal{F}_{\mathsf{TA}}^n$	yes	
none	$\mathcal{F}_{\mathsf{TA}}^n$	$\mathcal{F}_{\mathsf{TA}}^{n+1}$	no	Th. 3.5
none	$\mathcal{F}_{\mathsf{NTA}}$	$\mathcal{F}_{\mathsf{NTA}}$	no	Th. 3.5
$\mathcal{F}_{\mathsf{TA}}$	any	any	yes	Fig. 1, V receives messages from the prover's TA

4 Attacks Based on Public Key Registration Shift

ZK protocols have the property that a verifier cannot simulate a prover *after* completion of the attack. Nevertheless, the verifier could still play the *Mafia fraud* attack [9]. Indeed, if a third party, say Eve, and the verifier V^* are online, V^* may just relay messages between the prover and Eve while Eve may play a role of a honest verifier. This type of attack is addressed by a stronger notion of non-transferability. More concretely, we define non-transferability based on [20].

Definition 4.1 *Let O_P, O_V, O_V^*, O_W^* be some lists of oracles. Let (P^{O_P}, V^{O_V}) be an interactive argument of knowledge for a relation R in the key registration model. We assume that the verifier $\mathcal{P}_V$ is designated by a reference given as an extra common input. We say that the argument is non-transferable relative to $O_V^* | O_W^*$ if, for any malicious (polynomial-time) verifier $V^{*O_V^*}$ run by $\mathcal{P}_V$, and any polynomial-time $W^{*O_W^*}$ interacting with V^* and not run by $\mathcal{P}_V$, there exists a simulator $S^{O_V^*}$ run by $\mathcal{P}_V$ such that for any x, w and z such that $R(x,w)$ the random variables (over all random coins)*

$$\mathsf{View}_{W^*}(P^{O_P}(x,\mathcal{P}_V,w;r_P) \leftrightarrow V^{*O_V^*}(x,\mathcal{P}_V,z;r_V) \leftrightarrow W^{*O_W^*}(x;r_W))$$

and $\mathsf{View}_{W^*}(S^{O_V^*}(x,z;r) \leftrightarrow W^{*O_W^*}(x;r_W))$ are indistinguishable.

Thanks to Lemma 3.3, by using a dummy W^* receiving a single message y and doing nothing else we can see that for any O_W^*, non-transferability relative to $O_V^* | O_W^*$ implies O_V^*-zero-knowledge. Hence, non-transferability is a stronger notion than zero-knowledge (thus deniability).

Interestingly, the protocol of Fig. 1 using no key registration is non-transferable since for any V^* we can simulate the fact that the algorithm receives the string $[C:1,x]$ without the help of the prover. But maybe this is not the ideal non-transferable protocol that we want to use because it requires a secure channel from the prover's TA to the verifier. So, in what follows we assume that the prover uses no trusted agent.

A classical technique to achieve such a strong non-transferability uses proofs to a designated verifier V. This verifier is designated by its public key. That is, the verifier holds a public/private key pair (k,s). One way to make interactive proofs non-transferable consists of replacing the proof of knowledge for secret w by a proof of knowledge of either w or s. This way, a malicious verifier trying to transfer the proof to someone else will not prove knowledge of w since the verifier is assumed to hold s. This works because the verifier is not able to deny knowing s.

Practically, non-transferability strongly relies on the key setup assumption. To formalize this, we use a key registration model. If $\mathcal{P}$ wants to register a public key k,

he runs the $(\mathsf{Reg}, \mathsf{Dir})$ registration protocol with the registration authority. We model the key registration authority by a new functionality $\mathcal{F}_{\mathsf{CA}}^{\mathsf{Dir}}$ which registers $(\mathcal{P}, k)$ in a directory. We assume that this functionality is first set up with coins r_D. An instance of the functionality is referred to by sid.

Query $\mathsf{REGISTER}(\mathsf{sid})$ **from** $\mathcal{P}$**:** launch a Dir protocol session to interact with $\mathcal{P}$. If an output k is produced, store $(\mathcal{P}, k)$. Ignore any further $\mathsf{REGISTER}(\mathsf{sid})$ query.

Query $\mathsf{CHECK}(\mathsf{sid}, \mathcal{P}, k)$ **from** $\mathcal{P}'$**:** if $(\mathcal{P}, k)$ is stored, sends $(\mathsf{sid}, \mathsf{yes})$ to $\mathcal{P}'$. Otherwise sends $(\mathsf{sid}, \mathsf{no})$ to $\mathcal{P}'$.

In [22], Ristenpart and Yilek define several key registration models. They consider an arbitrary key registration protocol $(\mathsf{Reg}, \mathsf{Dir})$ in which $\mathsf{Reg}(k, s; r_R)$ is run by any registrant participant $\mathcal{P}$ with secret key s willing to register a public key k (e.g. generated by some algorithm G) and $\mathsf{Dir}(r_D)$ is run by a certificate authority which is assumed to be trusted. Several examples of key registration protocols are defined in [22]. The Plain protocol simply consists of sending a public key from Reg to Dir. The Kosk protocol (as for *Knowledge Of Secret Key*) consists of sending a public key joined with the secret key from Reg to Dir so that the directory authority can check that the registrant knows a secret key consistent with the registered public key. This quite strong model has a brother protocol consisting of making the authority generate the key pair and sending it to the registrant. This protocol is called KRK as for *Key Registration with Knowledge* in [1]. This is the model that could be used when discussing about identity-based protocols because of their intrinsic escrow property. The SPop protocol (as for *Signature-based Proof of Possession*) consists of sending a public key as a self-signed certificate. The Dir protocol first checks the signature before accepting the public key. This is indeed what is used in many practical cases to register a key in a Public Key Infrastructure (PKI). However, the SPop protocol is a pretty weak proof of possession while the Kosk protocol leads to important privacy concerns due to key escrow. We enrich this list with a ZKPop protocol which is an arbitrary zero-knowledge proof of knowledge for the secret key attached to the public key.

Our point is that if proving ignorance of s is doable for V^* then the transference attack could still apply with this construction. More formally, in a protocol where V receives messages from the prover and O_V only, with $O_W^* = O_V$, if V^* acts as a relay between the prover and W^* and is able to register a public key generated by W^* without knowing the secret key, then V^* literally transfers the proof to W^*. We have thus to check which registration model makes it possible for V^* to register a public key k while being able to prove ignorance of the secret key s. This is clearly the case of the Plain and ZKPop models: since V^* and W^* are colluding, V^* can just relay messages between the registering authority and W^* and learns nothing about s. In the SPop model (where V^* acting the same would have to learn more than the public key to register), we have to assume that a self-signed certificate does not provide any extra information to simulate a malicious prover to show that the proof is transfered. On the other side, this is clearly not the case of protocols based on key escrow such as the Kosk or KRK models. Indeed, key escrow surprisingly helps privacy by restoring non-transferability in the trusted agent model.

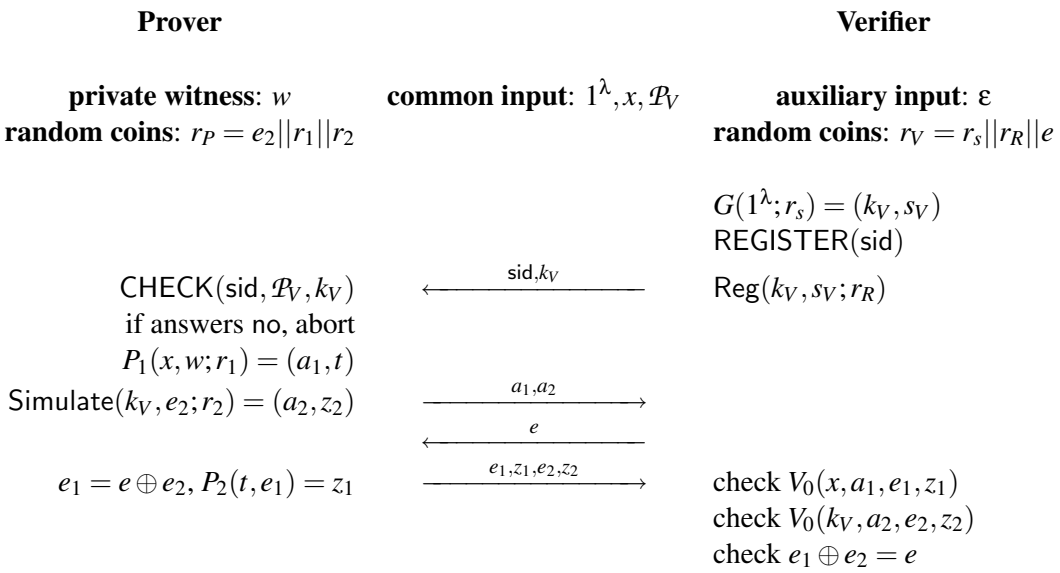

Figure 2: A non-transferable ZK proof system.

Theorem 4.2 *Let O_V^*, O_W^* be two lists of oracles. In the key registration model using* Kosk *there exists some argument of knowledge $(P, V^{\mathcal{F}_{\mathsf{CA}}^{\mathsf{Dir}}})$ which is non-transferable relative to $O_V^* | O_W^*$.*

Consequently, $(P, V^{\mathcal{F}_{\mathsf{CA}}^{\mathsf{Dir}}})$ is a O_V^*-ZK even when O_V^* includes trusted agents.

Proof: We use a Σ protocol defined by four algorithms $P_1(x, w; r_P) = (a, t)$, $P_2(t, e) = z$, $\mathsf{Extract}(x, a, e, z, e', z') = w$, and $\mathsf{Simulate}(x, e; r) = (a, z)$, a bitlength $\ell(\lambda)$ defining the domain for e, and a polynomially computable predicate $V_0(x, a, e, z)$. Following [7], the Σ protocol $P(x, w; r_P) \leftrightarrow V(x; r_V)$ works as follows: the prover runs $P_1(x, w; r_P) = (a, t)$ and sends a to V; The verifier picks a random bitstring e of $\ell(\lambda)$ bits and sends it to P; The prover runs $P_2(t, e) = z$ and sends z to V; The verifier accepts if and only if $V_0(x, a, e, z)$ holds. Following the definition of Σ-protocols, the verifier always accept if $R(x, w)$ holds and the protocol is correctly executed; $\mathsf{Extract}(x, a, e, z, e', z')$ must returns a witness w' such that $R(x, w')$ whenever the conditions $e \neq e'$, $V_0(x, a, e, z)$, and $V_0(x, a, e', z')$ are satisfied (this is the *special soundness* property); and for any x and e, $\mathsf{Simulate}(x, e; r) = (a, z)$ should define a random (a, e, z) triplet such that $V_0(x, a, e, z)$ holds with same distribution as the triplets generated by the honest run of the protocol (this is the *special zero-knowledge* property).

We modify the protocol of [8] as depicted on Fig. 2. Relative to the Kosk key registration model, obtain a non-transferable argument of knowledge. If V^* does not send any valid k_V binded to $\mathcal{P}_V$ to P, the simulation is trivial since it does not use w. Otherwise, V^* must have sent some k_V together with s_V to $\mathcal{F}_{\mathsf{CA}}^{\mathsf{Dir}}$. A simulator for V^* could then use s_V to simulate P in the OR proof. QED

QED

In this construction based on key escrow, the registering authority could abuse the protocol and make a cheating prover to the designated verifier. We further show that

this (bad) property is necessary for any $(P, V^{\mathcal{F}_{\mathsf{CA}}^{\mathsf{Dir}}})$ protocol which is non-transferable.

Theorem 4.3 *Let O_P, O_V be two lists of oracles. We assume that O_V is well-formed and that $\mathcal{P}_V$ can only receive messages from either O_V, $\mathcal{F}_{\mathsf{CA}}^{\mathsf{Dir}}$, or $\mathcal{P}_P$. Let $\mathcal{P}_D$ be an authority who runs an emulator D for the $\mathcal{F}_{\mathsf{CA}}^{\mathsf{Dir}}$ functionality for a given protocol $(\mathsf{Reg}, \mathsf{Dir})$. Let $(P^{O_P}, V^{O_V, \mathcal{F}_{\mathsf{CA}}^{\mathsf{Dir}}})$ be an argument of knowledge for R which is non-transferable relative to $\perp | O_V$. We let $\tilde{V}^{O_V}(x, \mathcal{P}_V, z; r_V)$ denote the protocol who simulates $V^{O_V \mathcal{F}_{\mathsf{CA}}^{\mathsf{Dir}}}(x, \mathcal{P}_V, z; r_V)$ with all messages for $\mathcal{P}_D$ and $\mathcal{P}_P$ sent to the same counterpart. There exists an algorithm $D^{*O_P}(x, z; r)$ such that for any r_I, x, z, $D^{*O_P} \leftrightarrow \tilde{V}^{O_P}$ accepts with high probability.*

This means that if an interactive proof using well-formed oracles for the verifier is non-transferable and with the property that the registering authority cannot cheat with the verifier, then the verifier $\mathcal{P}_V$ must receive messages from an oracle O_P, e.g. using a TA.

Proof: We let $W^{*O_V} = \tilde{V}^{O_V}$ be run by a participant $\mathcal{P}_W$ with $\mathcal{P}_V$ as counterpart. Here, V^* is "router" who "reroutes" the requests by W^* to the registration authority to $\mathcal{P}_D$ and others to $\mathcal{P}_P$. Clearly, $P^{O_P} \leftrightarrow V^{*\mathcal{F}_{\mathsf{CA}}^{\mathsf{Dir}}} \leftrightarrow W^{*O_V}$ makes W^* always accept since O_V is well-formed. Thanks to the definition of non-transferability, there is a simulator S such that the view from W^* to either $P^{O_P} \leftrightarrow V^{*\mathcal{F}_{\mathsf{CA}}^{\mathsf{Dir}}} \leftrightarrow W^{*O_V}$ or $S^{\mathcal{F}_{\mathsf{CA}}^{\mathsf{Dir}}} \leftrightarrow W^{*O_V}$ are indistinguishable. We can thus define $D^* = S^{\mathcal{F}_{\mathsf{CA}}^{\mathsf{Dir}}}$ so that $D^* \leftrightarrow \tilde{V}$ accepts with high probability. QED

QED

5 Malicious Applications

5.1 Shedding Light on Invisible Signatures (Invisibility Loss)

Undeniable signatures (aka *invisible signatures*) were invented by Chaum and van Antwerpen in [6] and have two basic features: (i) *interactive verification*, that is, the verification process is interactive and so the signer can choose who can verify his signature; (ii) *disavowal protocol* which allows the signer to prove that a given signature is a forgery. The first feature enables the signer to restrict the verification of the signature to those he wishes to. If the document leaks, a third party would not be able to verify the signature alone.

More formally, an invisible signature scheme is defined by two algorithms and a relation R: algorithm $\mathsf{Setup}(1^\lambda; K_s) = K_p$ is making keys and algorithm $\mathsf{Sign}(m, K_s; r) = s$ is making signatures. The relation $R(x, w)$ with witness $w = K_s$ defines valid signatures $x = (m, s, K_p)$. The scheme also comes with two ZK proof of knowledge protocols

$$(P_{\mathsf{Confirm}}(x, K_s; r_P), V_{\mathsf{Confirm}}(x; r_V)) \quad \text{and} \quad (P_{\mathsf{Deny}}(x, K_s; r_P), V_{\mathsf{Deny}}(x; r_V))$$

for the relations R and $\neg R$, respectively. Besides the zero-knowledge proof of knowledge properties, the scheme requires signature to be *existentially unforgeable* and *invisible*. Several definitions for invisibility exist in the literature. The weakest one requires the existence of a simulator $S(m, K_p; r) = s$ that makes strings look like signatures, such

that no algorithm based on K_p only can distinguish between Sign and S. This does not prevent from transferability issues. Clearly, a verifier V^* for (Confirm or Deny) equipped with a trusted agent O could transfer a proof universally from Lemma 3.4 to any offline W. Somehow, this malicious verifier would remove the "invisibility shield" on the signature which would then become visible.

There are some invisible signature schemes featuring non-transferability properties [20]. They however require verifiers to be given public and privates keys as well. We have seen how to defeat this protection in Section 4.

5.2 Selling Votes (Receipt-Freeness Loss)

Another application where transferring the protocol to a trusted agent would be dangerous is for e-voting: clearly, a trusted agent casting a ballot on behalf of a malicious elector could later testify the ballot content and receipt-freeness would be broken. E-democracy could collapse due to corruption with the help of trusted agents.

5.3 Denying Ring Signatures (Anonymity Loss)

Ring signatures were proposed by Rivest, Shamir and Tauman [23] to allow members of a certain group to sign a message without conveying any information on who inside the group signed the message. Informally, the ring signature works as follows. A signer creates a "ring" of members including himself. Each ring member $1 \leq i \leq n$ has a public k_i and secret key s_i. The public key specifies a trapdoor permutation and the secret key specifies the trapdoor information needed to compute its inverse. The signing process generates a ciphertext that could have been generated by anyone knowing at least one secret key. The verification process only requires the knowledge of the public keys. This way, the signer can hide in a ring that he created himself. Ring signature can be used e.g. for whistleblowing in order to protect the signer. Ring signatures can be used as a countermeasure to spamming. The idea is to have every email sent together with a ring signature of a ring consisting of the sender P and the receiver V. The reason to have email signed by the sender is to authenticate its origin and moreover, to make the email somehow binding. The reason to have the receiver in the ring is to prevent him from exhibiting the signed email to a third party W^*. In such a case, the email could have been forged by the receiver V^* so the sender can deny it.

If one member, say Victor, of the ring can prove the ignorance of his own secret key, then he can show that he was not able to sign any message with the ring signature, that is, he denies the signature. One way for Victor doing this in the Plain registration model is to take some pseudorandom generator π, some seed x and publish as public key $k = \pi(x)$. In this way he could present the pseudorandom generator and the seed to a third party W^* and convince him that he was not able to use the ring signature, since he did not know the secret key. To fix this, the key registration model should at least mandate the use of a proof of possession of a secret key, e.g. using self-signed certificates like the SPop or ZKPop protocol.

To defeat this, Victor owning a trusted agent could have his agent to simulate a key registration process so that only the agent would know the secret key. The attack could be more vicious here since the agent could still be used to sign messages in a ring. The only difference is that the agent would keep record of all signed messages

and could, upon request, certify that a message was signed or not. The signature by the trusted agent of the certificate together with its code is an evidence to anyone trusting the agent.

In [22], Ristenpart and Yilek proved that ring signatures could guaranty anonymity for rings larger than 2 even when the adversary can select the keys under the key registration model using an SPop protocol. Clearly, this result is no longer valid in the trusted agent model.

Once again, the attack relies on the public key registration issue, and the only way to thwart it seems to use key escrow: the Kosk protocols. This however enables the registration authority to forge a ring signature with a ring consisting of honest P and V: having V's secret key makes it possible to impersonate P to V. Finally, it seems that we either have to choose between having signatures deniable or forgeable.

We could still fix this problem by making honest participants use trusted agents to help registering keys in a Kosk-like model still ensuring privacy: a trusted agent could simulate Dir running Kosk. The certificate from the trusted agent could then be sent to D to register. Again, this fix is void if malicious registrants use nested trusted agents.

6 Conclusions

We have defined the Trusted Agent Model. In the past, several cryptographic protocols requiring trusted agents have been proposed but researchers prefer to develop protocols without them. However it does not prevent to *maliciously use* such devices if they exist. We devised scenarii in which adversaries equipped with such devices could defeat several cryptographic properties, e.g. invisibility in invisible signatures, receipt-freeness in e-voting, or anonymity in ring signatures. Fundamentally, these failures come from the strange nature of deniability in protocols. We have shown that deniability is not possible for regular protocols (namely, protocols not using trusted agents) if adversaries can use trusted agents. Deniability becomes possible again if honest and corrupted verifiers can use trusted agents. It collapses again if the malicious verifier can use a nested trusted agent of depth higher than the one the honest verifier is using. It can be restored again in a key registration model. We can even achieve non-transferability which is a stronger form of deniability, but this comes at the price of key escrow: if a protocol is non-transferable, then the key registration authority has the privilege to create malicious provers. Namely, non-transferability requires giving the authority some piece of information which could be used to cheat with proofs, which is pretty bad for privacy. An ultimate solution consists of making the proving part trivial by having proofs (resp. signatures) assessed by a trusted agent instead of running a protocol with the prover.

Although our "attacks" are pretty trivial, we think the issue of malicious use of trusted devices in practice has been overlooked so far. Clearly, these devices could defeat some privacy protocols. To the authors it does not seem acceptable on one hand, to accept tamper-proof hardware, and on the other hand, assume that adversaries cannot use such hardware and its properties to perform attacks.

Probably, the most interesting question that this paper opens is whether privacy is a self-contradicting concept or not. As shown herein, as soon as we place boundaries around devices, we can no longer control how private data spreads, so boundaries

are somehow harming privacy. On the other hand, privacy strongly relies on setting boundaries on data.

Acknowledgments

P. Mateus was partially supported by Instituto de Telecomunicações, FCT and EU FEDER through PTDC, namely via QSec PTDC/EIA/67661/2006 Project, IT project QuantTel, European Network of Excellence EURO-NF and IBM Portuguese scientific prize 2005.

References

[1] B. Barak, R. Canetti, J.B. Nielsen, and R. Pass. Universally composable protocols with relaxed set-up assumptions. In *Annual ACM Symposium on Theory of Computing: FOCS'04*, pages 186–195. IEEE Computer Society, 2004.

[2] J. Camenisch and M. Michels. Confirmer signature schemes secure against adaptive adversaries. In B. Preneel, editor, *Advances in Cryptology: EUROCRYPT'00*, pages 243–258. Springer, 2000.

[3] R. Canetti. Obtaining universally composable security: Towards the bare bones of trust. In K. Kurosawa, editor, *Advances in Cryptology - ASIACRYPT'07*, pages 88–112. Springer, 2007.

[4] R. Canetti and M. Fischlin. Universally composable commitments. In J. Kilian, editor, *Advances in Cryptology - CRYPTO'01*, pages 19–40. Springer, 2001.

[5] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *Annual ACM Symposium on Theory of Computing: STOC'02*, pages 494–503, New York, NY, USA, May 2002. ACM Press.

[6] D. Chaum and H. Van Antwerpen. Undeniable signatures. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 212–216. Springer, 1989.

[7] R. Cramer. Modular design of secure, yet practical cryptographic protocols. Phd thesis, University of Amsterdam, 1996.

[8] I. Damgård. On Σ-protocols. Lecture notes, University of Aahrus, 2005. Available at http://www.daimi.au.dk/%7Eivan/Sigma.pdf.

[9] Y. Desmedt. Major security problems with the unforgeable (feige)-fiat-shamir proofs of identity and how to overcome them. In *The 6th Worldwide Congress on Computer and Communications Security and Protection: Securicom'88*, pages 147–149. SEDEP, 1988.

[10] Y. Desmedt and J.-J. Quisquater. Public-key systems based on the difficulty of tampering. In Andrew M. Odlyzko, editor, *Advances in Cryptology: CRYPTO'86*, pages 111–117. Springer, 1987.

[11] Y. Desmedt and M. Yung. Weaknesses of undeniable signature schemes. In D.W. Davies, editor, *Advances in Cryptology: EUROCRYPT'91*, pages 205–220. Springer, 1991.

[12] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, STOC 85*, pages 291–304. ACM, 1985.

[13] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal of Computing*, 18(1):186–208, 1989.

[14] V. Gratzer and D. Naccache. Alien *vs.* quine, the vanishing circuit and other tales from the industry's crypt. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 48–58. Springer, 2006.

[15] M. Jakobsson. Blackmailing using undeniable signatures. In A. de Santis, editor, *Advances in Cryptology: EUROCRYPT'94*, pages 425–427. Springer, 1994.

[16] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In U. Maurer, editor, *Advances in Cryptology: EUROCRYPT'96*, pages 143–154. Springer, 1996.

[17] J. Katz. Universally composable multi-party computation using tamper-proof hardware. In M. Naor, editor, *Advances in Cryptology: EUROCRYPT'07*, pages 115–128. Springer, 2007.

[18] P. Mateus. Attacking zero-knowledge proof systems. Habilitation thesis, Department of Mathematics, Instituto Superior Técnico, 1049-001 Lisboa, Portugal, 2005. In Portuguese. Awarded the Portuguese IBM Scientific Prize 2005.

[19] P. Mateus, F. Moura, and J. Rasga. Transferring proofs of zero-knowledge systems with quantum correlations. In P. Dini et al, editor, *Proceedings of the First Workshop on Quantum Security: QSec'07*, page 0009. IEEE Press, 2007.

[20] J. Monnerat and S. Vaudenay. Short 2-move undeniable signatures. In *Proceedings of VietCrypt 06*, volume 4341, pages 19–36. Springer, 2006.

[21] R. Pass. On deniability in the common reference string and random oracle model. In D. Boneh, editor, *Advances in Cryptology: CRYPTO'03*, pages 316–337. Springer, 2003.

[22] T. Ristenpart and S. Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In *Advances in Cryptology: EUROCRYPT'07*, volume 4515, pages 228–245. Springer, 2007.

[23] R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565. Springer, 2001.

A Secure Commitment Using Trusted Agents

We consider the $\mathcal{F}_{\mathsf{com}}$ functionality[7] defined by

Query $\mathsf{COMMIT}(\mathsf{sid}, \mathcal{P}, \mathcal{P}', b)$ **from** $\mathcal{P}$**:** record b, send the message $[\mathsf{receipt}, \mathsf{sid}, \mathcal{P}, \mathcal{P}']$ to $\mathcal{P}'$, and ignore any future COMMIT queries.

Query $\mathsf{OPEN}(\mathsf{sid})$ **from** $\mathcal{P}$**:** if no value b was recorded, ignore. Otherwise, send $[\mathsf{open}, \mathsf{sid}, b]$ to $\mathcal{P}'$.

Here is a protocol to realize this ideal functionality using trusted agents:

- To emulate $\mathsf{COMMIT}(\mathsf{sid}, \mathcal{P}, \mathcal{P}', b)$ by $\mathcal{P}$, participant $\mathcal{P}$ sets a code C as detailed below, makes the queries $\mathsf{SEND}(\mathsf{sid}, C)$ to $\mathcal{F}_{\mathsf{TA}}$, then $\mathsf{SEND}(\mathsf{sid}, b)$, then $\mathsf{SHOWTO}(\mathsf{sid}, \mathcal{P}')$. Participant $\mathcal{P}'$ gets a message $[C : \mathsf{receipt}, N]$, checks that it comes from $\mathcal{F}_{\mathsf{TA}}$, that C is correct (as defined below), and stores N.
- To emulate $\mathsf{OPEN}(\mathsf{sid})$, $\mathcal{P}$ queries $\mathsf{SEND}(\mathsf{sid}, \emptyset)$ to $\mathcal{F}_{\mathsf{TA}}$ and finally $\mathsf{SHOWTO}(\mathsf{sid}, \mathcal{P}')$. Participant $\mathcal{P}'$ gets a message $[C : \mathsf{open}, N', b']$, checks that it comes from $\mathcal{F}_{\mathsf{TA}}$, that C is still correct, and that $N = N'$. The value b' is revealed.

The code C takes a first message b as input, picks a random nonce N, stores N and b, and responds by $(\mathsf{receipt}, N)$. Then it waits for a dummy message and responds by (open, N, b).

Given a static adversary $\mathcal{A}$ interacting with $\mathcal{P}$, $\mathcal{P}'$ and $\mathcal{F}_{\mathsf{TA}}$ in the real world, we define a simulator $\mathcal{S}$ interacting with $\mathcal{P}$, $\mathcal{P}'$ and $\mathcal{F}_{\mathsf{TA}}$ in the ideal world so that for any environment $\mathcal{E}$ interacting with $\mathcal{P}$, $\mathcal{P}'$ and $\mathcal{A}$ resp. $\mathcal{S}$, the real and ideal views of $\mathcal{E}$ are indistinguishable. Therefore, the protocol UC-realizes $\mathcal{F}_{\mathsf{com}}$ in the $\mathcal{F}_{\mathsf{TA}}$-hybrid model.

Indeed, when $\mathcal{P}$ and $\mathcal{P}'$ are both honest or both corrupted, constructing the simulator is trivial. When $\mathcal{P}$ is honest but $\mathcal{P}'$ is corrupted, the $[C : \mathsf{receipt}, N]$ message to $\mathcal{P}'$ can be perfectly simulated from the $[\mathsf{receipt}, \mathsf{sid}, \mathcal{P}, \mathcal{P}']$ by picking a fresh nonce N and maintain a table of $\mathsf{sid} \leftrightarrow N$ pairs. The $[C : \mathsf{open}, N', b']$ message can be simulated upon message $[\mathsf{open}, \mathsf{sid}, b]$ from $\mathcal{F}_{\mathsf{com}}$ by looking up at the table for the correct N' and setting $b' = b$. Thanks to the trusted agent property and the C code there must be an appropriate pair. When $\mathcal{P}$ is corrupted and $\mathcal{P}'$ is honest, the messages to $\mathcal{F}_{\mathsf{TA}}$ with correct code C can be perfectly simulated from the message $\mathsf{COMMIT}(\mathsf{sid}, \mathcal{P}, \mathcal{P}', b)$ resp. $\mathsf{OPEN}(\mathsf{sid})$ to $\mathcal{F}_{\mathsf{com}}$.

[7] The functionality $\mathcal{F}_{\mathsf{mcom}}$ [4] could be used as well.