On Privacy Losses in the Trusted Agent Model (Abstract)

Paulo Mateus[1] and Serge Vaudenay[2]

[1] SQIG /IT - DM/IST TULisbon
1049-001 Lisboa, Portugal
http://sqig.math.ist.utl.pt

[2] EPFL
CH-1015 Lausanne, Switzerland
http://lasecwww.epfl.ch

Abstract. Tamper-proof devices are pretty powerful. They typically make security applications simpler (provided that the tamper-proof assumption is not violated). For application requiring privacy, we observe that some properties may become harder (if possible at all) to achieve when devices are maliciously used. We take the example of deniability, receipt-freeness, and anonymity.

We formalize the *trusted agent model* which assumes tamper-proof hardware in a way which captures the notion of programmable secure hardware. This model defines a functionality relative to which deniability requires provers to use a tamper proof hardware. Otherwise, any asymmetric situation in which the malicious verifiers have more powerful tamper-proof devices than the honest ones makes deniability impossible.

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 some privacy notions.

1 Introduction

Since it is a good idea to ground the security of a system by assuming a powerful adversary, we should not exclude adversaries which would *deliberately* cast themselves in a restrictive computation model, where they have no full control over their memory and computation. Unfortunately, this is the assumption made by several cryptographic schemes, such as undeniable [5] and ring signatures [18]. The idea of using tamper-proof devices to realize cryptographic functionalities goes back (at least) to 1986 [6]. Here, we consider using tamper-proof devices for malicious purposes. In this paper we show how to make several privacy attacks using trusted tamper-proof devices.

An interactive zero-knowledge (ZK) proof system [8] is a two-party protocol, between a prover and a verifier, where the prover is able to convince the verifier that a statement holds, without conveying any information that could not be obtained without interacting with the prover. Classical ZK proof systems fulfill a privacy property called *deniability* [16] 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 undeniable signatures [5] 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 [18] 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.

Tamper-proof devices have been massively used 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. While these devices exist mostly to defend against privacy attacks, we show that they can also be used to perform them. We do so 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. We show that within this model and an appropriate universal program, it is possible

– to transfer proofs of zero-knowledge protocols after completion (in particular: to transfer the verification of an undeniable 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, a tamper-proof device implementing a trusted agent can be used to verify a proof. Afterward, it can testify that the verification protocol was correctly run by some kind of forensic evidence. Clearly, such a device could be used to kill the undeniable signature paradigm [5]. One could say that this trivial attack could be defeated by classical non-transferability techniques like having a PKI for verifiers [3,7,10,11]. However, the cryptographic assumption of this approach highly depends on the public-key registration process and happens to be insufficient when we cannot prevent the malicious verifier for holding the tamper-proof device during the key registration phase, unless the secret key is escrowed. Key escrow however leads us to other privacy concerns.

Related work. The idea of using tamper-proof hardware to transfer proofs of ZK protocols was first introduced in the context of quantum memory [13,14]. However, as discussed herein, it can be extended to commercially available hardware in a practical way. 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. These 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 general, setup phases in cryptographic protocols is a critical issue. Authors often assume that participants securely register their public keys in a honest manner, although doing so is not trivial. Key setup is still a problem for the Universal Composability (UC) framework by Canetti [4]. For instance, the key registration model for UC framework by Barak, Canetti, Nielsen and Pass [1] is quite minimal but assumes that the secret key of honest participants is safely stored by the key registration authority (so that even the participants ignore their keys). In [17], 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 [12] defined the notion of tamper-proof hardware token. 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 still running in a state which is a consequence of having set it up with this code.

In [16], Pass introduced the notion of deniable zero-knowledge which is immune to offline proof transfer. Plain zero-knowledge in the standard model is essentially deniable. However, zero-knowledge in the common reference string (CRS) model is not deniable when it really requires CRS to work [16]. For instance, non-interactive zero-knowledge [2] is clearly not deniable but can be achieved in the CRS model. Deniable zero-knowledge in the random oracle model can be achieved [16]. Our trusted agent model is at least as strong as the random oracle model and shares similar properties: regular zero-knowledge protocols collapse when the malicious verifier uses trusted agents. It can be restored by having honest verifiers to use trusted agents or an appropriate key registration model. However, if trusted agents can be *placed inside* other trusted agents, then only a registration model with key escrow will restore deniability.

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.

2 The Trusted Agent Model

In a multiparty setting, several participants or functionalities 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 $\mathcal{O}$ will be referred to $\mathcal{P}$ querying an *oracle* $\mathcal{O}$ and we will write $\mathcal{P}^{\mathcal{O}}$ 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 $\mathcal{O}_P$ (resp. $\mathcal{O}_V$) is the list of functionalities that participant $\mathcal{P}_P$ (resp. $\mathcal{P}_V$) may invoke when running the algorithm P (resp. V) we denote by $P^{\mathcal{O}_P} \leftrightarrow V^{\mathcal{O}_V}$ the interaction. More precisely, we denote by $P^{\mathcal{O}_P(r_{OP})}(x_P; r_P) \leftrightarrow V^{\mathcal{O}_V(r_{OV})}(x_V; r_V)$ the experiment of running P with input x_P and random coins r_P with access to $\mathcal{O}_P$ initialized with random coins r_{OP} and interacting with V with input x_V and random coins r_V with access to $\mathcal{O}_V$ initialized with random coins r_{OV}. We denote by $\mathsf{View}_V(P^{\mathcal{O}_P(r_{OP})}(x_P; r_P) \leftrightarrow V^{\mathcal{O}_V(r_{OV})}(x_V; r_V))$ the *final view* of V in this experiment, i.e. x_V, r_V and the list of messages from either P or $\mathcal{O}_V$.

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). 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.

We consider (probabilistic) interactive Turing machines with four kinds of tapes: the input tape, the working tape, the output tape, and the random tape. Their *state* is the state of the automaton and the content of the working tape. We consider a programming language to specify 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 values of $(\mathsf{sid}, \mathcal{P}, r, C)$ 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 (C, out) to $\mathcal{P}'$.

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 protocol context.

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

1. $\mathcal{P}_0$ runs $M^{\mathcal{O}}$ 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 $S^{\mathcal{O}}$.

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 $\mathcal{O}$ 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 $\mathcal{O}$. 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 trusted agents, they do check that messages come from the same holder but his identity has no influence on the result.

Clearly, our model for trusted agent could easily be implemented (assuming that tamper-proof devices could be manufactured) provided that we could trust the trusted agent manufacturer and that nobody knows how to counterfeit it. Obviously this is a quite strong assumption but this would still make sense for applications in which there is a liable entity which must be trusted. For instance, digital payment could rely on trusted agents issued by a liable bank. Indeed, most of customers now have credit cards with a tamper-proof embedded chips. Some provide trusted secure tokens like secureID to display private access code for e-banking. Global (nation-wide) e-governance could be based on protocols using trusted agents issued under the control of observers appointed by the congress. They already issue 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.

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

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

We recall here the definition of a zero-knowledge argument of knowledge system [8,9]. We adapt it so that the prover and the verifier can talk to a list of oracles (typically: trusted agents). Quite importantly, we do not assume that honest and malicious verifiers can use the same oracles.

Definition 1 (Zero-knowledge argument). *Let $R(x, w)$ be a predicate relative to a* statement *x and a* witness *w, $\mathcal{O}_P, \mathcal{O}_V, \mathcal{O}_V^*$ be three lists of oracles initialized using a list of random coins r_I. An* argument of knowledge *for R relative to $(\mathcal{O}_P, \mathcal{O}_V)$ is a pair $(P^{\mathcal{O}_P}, V^{\mathcal{O}_V})$ of ppt interactive machines $P^{\mathcal{O}_P}(x, w; r_P)$ and $V^{\mathcal{O}_V}(x, z; r_V)$ 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*: $\forall r_I, r_P, r_V, x, w, z \quad R(x, w) \Longrightarrow \left(P^{\mathcal{O}_P}(x, w; r_P) \leftrightarrow V^{\mathcal{O}_V}(x, z; r_V) \text{ accepts}\right)$*
- Soundness*: there exists a ppt algorithm E (called* extractor*) which is given full black-box access to the prover such that for any x and z, any ppt algorithm P^* with access to $\mathcal{O}_P$, if the probability (over all random coins) that $P^{*\mathcal{O}_P}(x; r_P) \leftrightarrow V^{\mathcal{O}_V}(x, z; r_V)$ 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 $\mathcal{O}_V^*$ (or $\mathcal{O}_V^*$-ZK) if for any ppt algorithm $V^{*\mathcal{O}_V^*}$ with access to $\mathcal{O}_V^*$ there exists a ppt algorithm $S^{\mathcal{O}_V^*}$ (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 $\mathsf{View}_V(P^{\mathcal{O}_P}(x, w; r_P) \leftrightarrow V^{*\mathcal{O}_V^*}(x, z; r_V))$ or running $S^{\mathcal{O}_V^*}(x, z; r)$ produce two random (over all random coins) outputs with indistinguishable distributions.*

Our definition of zero-knowledge is essentially *deniable* because the simulator can be run by the verifier [16]: V^* cannot produce some y which could serve to feed a malicious prover P^*.[3]

Theorem 2. *Let $\mathcal{O}_P, \mathcal{O}_V$ be any oracle lists. We assume $\mathcal{O}_V$ is well-formed. Let $N^{\mathcal{O}_V}$ be a nested trusted agent with $\mathcal{O}_V$ embedded. Let R be any hard predicate. No argument of knowledge $(P^{\mathcal{O}_P}, V^{\mathcal{O}_V})$ for R such that V only receives messages from $\mathcal{O}_V$ or the prover $\mathcal{P}_P$ is $N^{\mathcal{O}_V}$-ZK.*

In particular, if $\mathcal{O}$ is a trusted agent, no (P, V) argument for R is $\mathcal{O}$-ZK: if a malicious verifier can use a trusted agent but the honest verifier does not, the argument system is not deniable.

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.

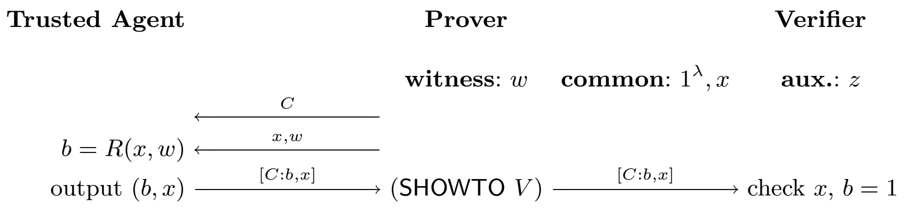

Fig. 1. UC-realization of the $\mathcal{F}_{\mathsf{ZK}}$ in the $\mathcal{F}_{\mathsf{TA}}$-hybrid model.

The table below summarizes feasibility results for ZK arguments depending on oracle accesses.

oracle for P	oracle for V	oracle for V^*	feasibility	comment
none	none	none	yes	
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. 2
none	$\mathcal{F}_{\mathsf{NTA}}$	$\mathcal{F}_{\mathsf{NTA}}$	no	Th. 2
$\mathcal{F}_{\mathsf{TA}}$	any	any	yes	Fig. 1

4 Malicious Use of Tamper-Proof Agents

Shedding light on invisible signatures (invisibility loss). Undeniable signatures were invented by Chaum and van Antwerpen in [5]. These are 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 ($\mathsf{Confirm}$ or Deny) equipped with a trusted agent $\mathcal{O}$ could transfer a proof universally to any offline W. Somehow, this malicious verifier would remove the "invisibility shield" on the signature which would then become visible.

There are some undeniable signature schemes featuring non-transferability properties [15]. They however require verifiers to be given public and privates keys as well.

[3] 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.

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.

Resurrecting rogue-key attacks on group signatures. Ring signatures were proposed by Rivest, Shamir and Tauman [18] 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. This is the rogue key attack. To defeat this, public keys must be securely registered, meaning that registrants should *prove* their knowledge of the secret key to the registration authority. In [17], 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 simple proof protocol consisting of issuing a self-signed certificate.

To defeat this countermeasure, 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.

Finally, we could prove that the only ways to avoid this new attack would be one of the following: to enforce key escrow by the authority; to make honest participants use trusted agents to help registering keys.

5 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 undeniable signatures, receipt-freeness in e-voting, or anonymity in ring signatures. Fundamentally, these failures come from the strange nature of deniability in protocols.

Although our attacks are pretty trivial, we think the issue of malicious use of trusted devices has been overlooked so far. Clearly, these devices could break some 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.

Acknowledgments

The work was partially supported by IBM Portugal, IT project, EU FEDER and FCT, namely via project Quantlog POCI/MAT/55796/2004 and project QSec PTDC/EIA/67661/2006, and by the European Network of Excellence initiative in Future Networks.

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. M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC'88*, pages 103–112. ACM, 1988.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal of Computing*, 18(1):186–208, 1989.
10. M. Jakobsson. Blackmailing using undeniable signatures. In A. de Santis, editor, *Advances in Cryptology: EUROCRYPT'94*, pages 425–427. Springer, 1994.
11. 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.
12. 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.
13. 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.
14. 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.
15. J. Monnerat and S. Vaudenay. Short 2-move undeniable signatures. In *Proceedings of VietCrypt 06*, volume 4341, pages 19–36. Springer, 2006.
16. 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.
17. 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.
18. 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.