On accountable certificate management

Peeter Laud*

joint work with

Ahto Buldas, Helger Lipmaa,

Meelis Roos, Jan Willemson

*Universität des Saarlandes

Tartu Ülikool

Küberneetika AS

We have encryption schemes, signature schemes, etc.

Public key systems: there are public keys and private keys.

How do I find the public signature key K_A of an entity named A?

- If A is known to me, then meet her in person and ask.
- Otherwise find someone ("CA") to introduce A to me.
  In this case I have to trust the CA in some aspects:
  - The key K_A that I received from CA must indeed belong to A.

How do I know that I received K_A from CA?

- Let CA send me $(\stackrel{K_A}{\rightarrow} A, \mathbf{sign}_{K_{CA}}(\stackrel{K_A}{\rightarrow} A))$.

I have a message M and signature $\mathbf{sign}_{K_A}(M)$. How do I convince a person B, that A signed that message?

I.e. how do I convince B that A used the public key K_A when she signed M?

- Present him with $(\stackrel{K_A}{\to} A, \mathbf{sign}_{K_{CA}}(\stackrel{K_A}{\to} A))$.

  For this to work, B also has to trust that CA gave out A's public key.

The question, whether $\stackrel{K_A}{\to} A$ holds, lies outside cryptography.

Thus CA's sayings *define* $\stackrel{\cdot}{\to}\cdot$ for cryptographic purposes.

An axiom from an authentication logic:
P **controls** $X \wedge P$ **says** $X \Rightarrow X$.

CA has to be trusted in certain aspects. In which aspects exactly?

Usually the following assumption is made:
$\forall P \forall K : \; CA$ **controls** $\stackrel{K}{\to} P$.

Thus CA **says** $\stackrel{K_A}{\to} A \land CA$ **says** $\stackrel{K_A}{\to} B$ is trusted not to happen.

(Define the semantics of $\stackrel{K}{\to} A$ in such way that it is a partial function from the set of keys to the set of principals.)

If someone obtains $(\stackrel{K_A}{\to} A, \mathbf{sign}_{K_{CA}}(\stackrel{K_A}{\to} A))$ and $(\stackrel{K_A}{\to} B, \mathbf{sign}_{K_{CA}}(\stackrel{K_A}{\to} B))$, then he can prove, that CA has abused the trust.

We say, that CA **says** $X_1 \land \cdots \land CA$ **says** X_n, where $X_1, \ldots, X_n \models \bot$, is a *fraud*.

We say that a [certificate management] system is *accountable*, if

- **Fraud detection:** Any interested party can detect and prove (to another party), whether a trusted party has commited any frauds.
- **Anti-framing:** An honest trusted party can explicitly disavow any false accusations against her.

For the current system (where CA makes statements of the form $(\stackrel{K_A}{\to} A, \mathbf{sign}_{K_{CA}}(\stackrel{K_A}{\to} A))$, when asked):

- Fraud detection is possible, if everybody hears everything that CA says.
- Anti-framing: if $(\stackrel{K_A}{\to} A, \mathbf{sign}_{K_{CA}}(\stackrel{K_A}{\to} A))$, $(\stackrel{K_A}{\to} B, \mathbf{sign}_{K_{CA}}(\stackrel{K_A}{\to} B))$ is presented to a CA, who didn't sign them, then
  - K_{CA}^{-1} has been stolen. A non-case.
  - Crypto has been broken (the signature scheme). To prove his innocence, CA must be able to show it (break it as well).

Thus the system isn't accountable.

We say that a formula X is *published*, if $(\forall P : \ P \ \mathbf{sees} \ X)$ holds.

We consider publication to be a protocol primitive.

An accountable system: let S be the [finite] set of all claims, that CA makes about the state of the world.
The protocol:

0. Preliminary step: CA publishes $(S, \mathbf{sign}_{K_{CA}}(S))$.

1. If a principal wants to make a query, then he looks up the publication. Also checks the signature.

Problem: efficiency. Very high communication costs.

A *collision-resistant* (CR) *hash function* is a function (algorithm) $H : \{0,1\}^* \longrightarrow \{0,1\}^k$, such that it is “very hard” to find such x_1, x_2, that $x_1 \neq x_2$ and $H(x_1) = H(x_2)$.

Next protocol:

0. Preliminary step: CA publishes $(H(S), \mathbf{sign}_{K_{CA}}(H(S)))$.
1. If a principal wants to make a query, then he notifies the CA. Also checks the signature.
2. The CA responds with S.
3. The principal verifies $H(S)$.

Still high communication costs.

Let's change the model a bit. Suppose, that

- the "real world" defines a finite set of bit-strings S;
- $\forall x \in \{0,1\}^* :\ CA$ **controls** $x \in S$;
- users query the CA, asking whether a certain bit-string is in S.

An efficient, non-accountable example: let (D, P, V) be an efficient CR accumulator. The protocol:

0. preliminary: CA publishes $(D(S), \mathbf{sign}_{K_{CA}}(D(S)))$;
1. user sends to CA: x?;
2. CA returns $P(x, S)$ or $((\mathsf{no}, x), \mathbf{sign}_{K_{CA}}(\mathsf{no}, x))$;
3. if $P(x, S)$ sent, then the user verifies $V(x, D(S), P(x, S))$.

An *attester* is a triple of algorithms (D, P, V), where

- $D : \mathcal{P}_f(\{0,1\}^*) \longrightarrow \{0,1\}^k$ is the digest algorithm.
- The proof algorithm P takes $S \subset_f \{0,1\}^*$ and $x \in \{0,1\}^*$ and produces the "inclusion/exclusion proof" $P(x, S) \in \{0,1\}^*$.
- The verification algorithm takes a string $x \in \{0,1\}^*$, a digest $d \in \{0,1\}^k$ and a proof $p \in \{0,1\}^*$ and returns either yes, no, or error, so that

$$\forall S \subset_f \{0,1\}^* \; \forall x \in S : V(x, D(S), P(x, S)) = \mathsf{yes}$$
$$\forall S \subset_f \{0,1\}^* \; \forall x \in \{0,1\}^* \backslash S : V(x, D(S), P(x, S)) = \mathsf{no}.$$

An attester is *undeniable* if it is "very hard" to find such $x, d, p, p' \in \{0,1\}^*$, that $V(x, d, p) = \mathsf{yes}$ and $V(x, d, p') = \mathsf{no}$.

Efficient and accountable example: let (D, P, V) be an undeniable attester,

0. preliminary: CA publishes $(D(S), \mathbf{sign}_{K_{CA}}(D(S)))$;
1. the user sends to CA: x?;
2. CA returns $P(x, S)$;
3. the user verifies $V(x, D(S), P(x, S))$.

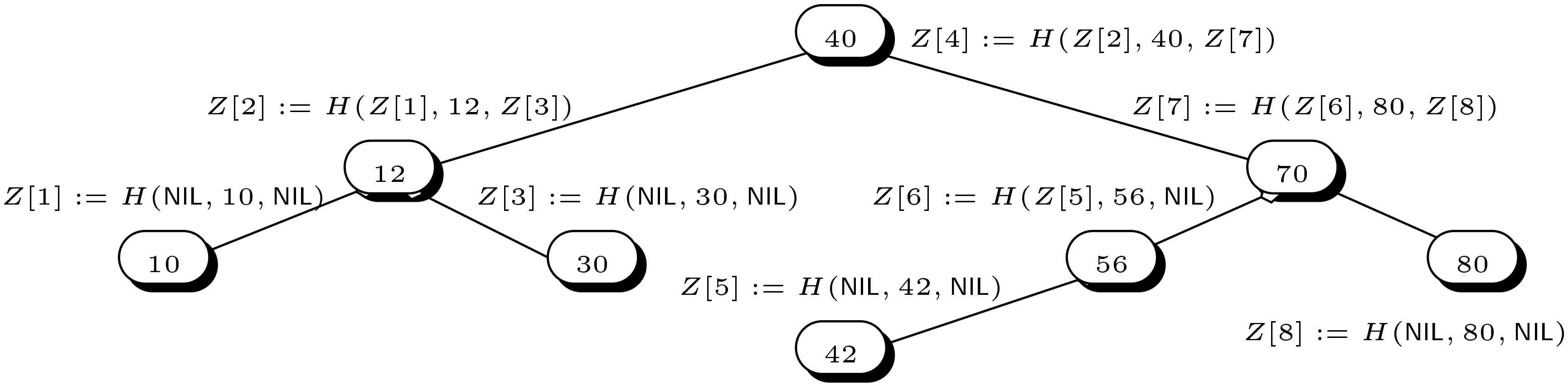
Fraud detection Producing a lying proof requires breaking the crypto, as it is easy to construct a true proof.

Anti-framing The accusation (an element with two contradictory proofs) together with $D(S)$ constitute a break.

An *authenticated search tree* for the set S consists of:

- a binary search tree with $|S|$ vertices;
  - vertices are labeled with elements of S;
  - denote the label of vertex v by $K[v]$;
- a second set of vertex labels $Z[v] := H(Z[v_L], K[v], Z[v_R])$, where
  - v_L and v_R are v-s left and right child;
  - the label of a missing vertex is denoted by NIL.

$S = \{10, 12, 30, 40, 42, 56, 70, 80\}$.



$D(S) = Z[4]$

$P(42, S) = P(43, S) = ((\mathsf{NIL}, 42, \mathsf{NIL}), [(56, \mathsf{NIL}), (70, Z[8]), (40, Z[2])])$

Model change:

- $S \subset_f \{0,1\}^* \times \{0,1\}^*$, S is a partial function;
- Users ask CA to evaluate the function at certain arguments.

Sort S by first component, let T be an auth. search tree for S.

0. CA publishes $(Z[\mathrm{root}_T], \mathbf{sign}_{K_{CA}}(Z[\mathrm{root}_T]))$;
1. the user sends to CA: $x?$;
2. if $\exists y : (x, y) \in S$:
   - let z_{pred} and z_{succ} be the elements of S immediately preceding and succeeding (x, y).
   - CA answers with $(y, P(z_{\mathrm{pred}}, S), P((x, y), S), P(z_{\mathrm{succ}}, S))$;

   otherwise CA answers with $P((x, \epsilon), S)$;
3. the user verifies proofs. Also verifies the location of z_{pred} and z_{succ}.

The database S might have quite an exotic structure with similarly exotic constraints on it.

- This may happen if the CA also issues attribute certificates.

The same holds true for the queries that users can make.

Things to do:

- Clarify, for which structures of S and queries there exist undeniable "attesters" allowing to answer those queries accountably.

- Better understand the connection between "real world" and crypto.