

Privacy-Compliant Authentication Protocols In Constrained Environments

Gildas Avoine

MIT, USA

<http://www.avoine.net>



Authentication? Er... What's that?

Constrained Environments, Example RFID

Efficient Challenge-Response Protocols

Other (Funny) Problems in Constrained Environments

Authentication? Er... What's that?

Authentication in Daily Lives



Proving to somebody else (**verifier**) that you are the person you claim to be (**prover**).

Verifier

Prover

← Identifier, Password

Do not resist to replay attacks!

Authentication by Challenge/Response

Verifier

Pick Challenge

Check Response

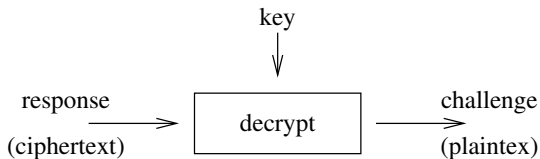
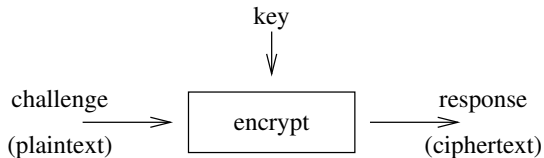
Challenge

Identifier, Response

Prover

Compute Response

How to Compute/Check the Response



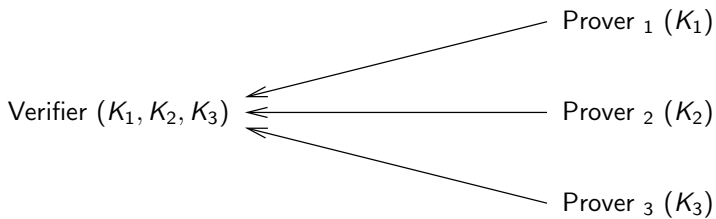
Keys to encrypt and decrypt are the same:

Symmetric cryptography aka secret-key cryptography

Keys are different (mathematically linked):

Asymmetric cryptography or **public-key cryptography**

(Based on mathematical functions, heavy)



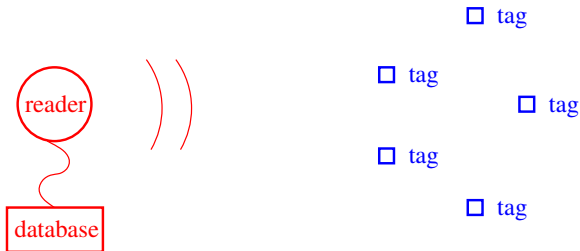
- Authentication **without revealing the identifier**.

- Authentication **without being tracked**.

(Given a set of interactions between provers and a verifier, an adversary must not be able to find any correlation in this set)

Constrained Environments, Example RFID

RFID is a technology to identify (**authenticate** in our case) an object or a person, remotely, using a small transponder (**tag**) that is attached or incorporated to the object or person to identify.



Willingness to develop **small** and **cheap** RFID tags

- Tags are passive
- Small Memory
- No public-key cryptography
- Not tamper-resistant
- Communication range is centimeters to meters
- Tags answer without the agreement of their bearers
- Tags cannot be switched-off

Authentication Requirements

- **Secure** i.e. prover authenticated iff prover knows right key
- Resistant to **compromized keys**
- **Efficient** i.e. reasonable amount of computation (prov/verif)

Authentication Requirements

- **Secure** i.e. prover authenticated iff prover knows right key
- Resistant to **compromized keys**
- **Efficient** i.e. reasonable amount of computation (prov/verif)
- **Lightweight** i.e. implementable on low-capabilities provers

Authentication Requirements

- **Secure** i.e. prover authenticated iff prover knows right key
- Resistant to **compromized keys**
- **Efficient** i.e. reasonable amount of computation (prov/verif)
- **Lightweight** i.e. implementable on low-capabilities provers
- Resistant to **traceability**

Authentication Requirements

- **Secure** i.e. prover authenticated iff prover knows right key
- Resistant to **compromized keys**
- **Efficient** i.e. reasonable amount of computation (prov/verif)
- ~~Lightweight i.e. implementable on low capabilities provers~~
- Resistant to **traceability**

Authentication Requirements

- **Secure** i.e. prover authenticated iff prover knows right key
 - Resistant to **compromized keys**
 - **Efficient** i.e. reasonable amount of computation (prov/verif)
 - **Lightweight** i.e. implementable on low-capabilities provers
 - Resistant to **traceability**
-
- Public-key encryption that is lightweight

Authentication Requirements

- **Secure** i.e. prover authenticated iff prover knows right key
 - Resistant to **compromized keys**
 - **Efficient** i.e. reasonable amount of computation (prov/verif)
 - **Lightweight** i.e. implementable on low-capabilities provers
 - ~~Resistant to traceability~~
-
- Public-key encryption that is lightweight

Authentication Requirements

- **Secure** i.e. prover authenticated iff prover knows right key
 - Resistant to **compromized keys**
 - **Efficient** i.e. reasonable amount of computation (prov/verif)
 - **Lightweight** i.e. implementable on low-capabilities provers
 - Resistant to **traceability**
-
- Public-key encryption that is lightweight
 - Private authentication protocols

Authentication Requirements

- **Secure** i.e. prover authenticated iff prover knows right key
 - Resistant to **compromized keys**
 - ~~Efficient i.e. reasonable amount of computation (prov/verif)~~
 - **Lightweight** i.e. implementable on low-capabilities provers
 - Resistant to **traceability**
-
- Public-key encryption that is lightweight
 - Private authentication protocols

Authentication Requirements

- **Secure** i.e. prover authenticated iff prover knows right key
 - Resistant to **compromized keys**
 - **Efficient** i.e. reasonable amount of computation (prov/verif)
 - **Lightweight** i.e. implementable on low-capabilities provers
 - Resistant to **traceability**
-
- Public-key encryption that is lightweight
 - Private authentication protocols
 - Efficient challenge-response protocols

Authentication Requirements

- **Secure** i.e. prover authenticated iff prover knows right key
 - Resistant to **compromized keys**
 - **Efficient** i.e. reasonable amount of computation (prov/verif)
 - **Lightweight** i.e. implementable on low-capabilities provers
 - Resistant to **traceability**
-
- Public-key encryption that is lightweight
 - Private authentication protocols
 - **Efficient challenge-response protocols**

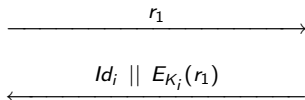
Efficient Challenge-Response Protocols

Private Challenge/Response Protocol

Verifier (Id_i, K_i)

database	
Id_1	K_1
Id_2	K_2
\vdots	\vdots
Id_i	K_i
\vdots	\vdots
Id_n	K_n

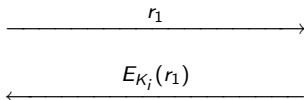
Prover (Id_i, K_i)



Verifier (Id_i, K_i)

database	
Id_1	K_1
Id_2	K_2
\vdots	\vdots
Id_i	K_i
\vdots	\vdots
Id_n	K_n

Prover (Id_i, K_i)



- The verifier must **check every key** of the database (exhaustive search).

Verifier (Id_i, K_i)

Prover (Id_i, K_i)

database	
Id_1	K_1
Id_2	K_2
\vdots	\vdots
Id_i	K_i
\vdots	\vdots
Id_n	K_n

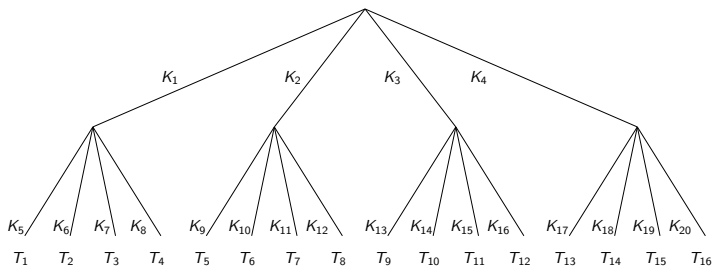
$\xrightarrow{r_1}$

$\xleftarrow{E_{K_i}(r_1, r_2)}$ pick r_2

- The verifier must **check every key** of the database (exhaustive search).
- The prover must **randomize** his response.

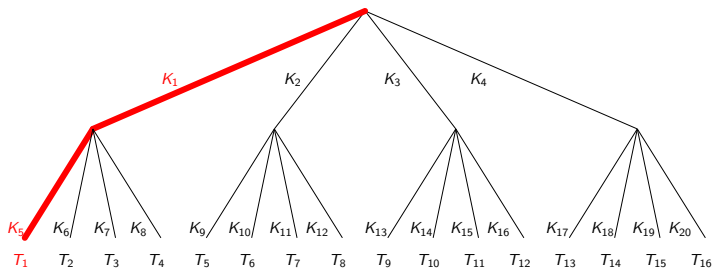
Molnar and Wagner Tree-Based Authentication

- Molnar and Wager suggested a tree-based technique to reduce the complexity from $O(n)$ to $O(\log n)$.
- The set of tags is recursively divided into **subgroups**. Each subgroup has its own key.



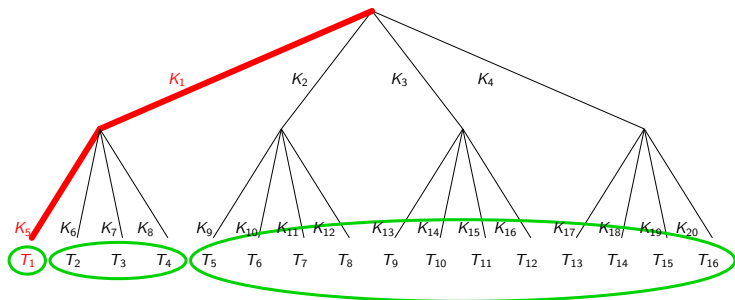
Molnar and Wagner Tree-Based Authentication

- Molnar and Wager suggested a tree-based technique to reduce the complexity from $O(n)$ to $O(\log n)$.
- The set of tags is recursively divided into **subgroups**. Each subgroup has its own key.

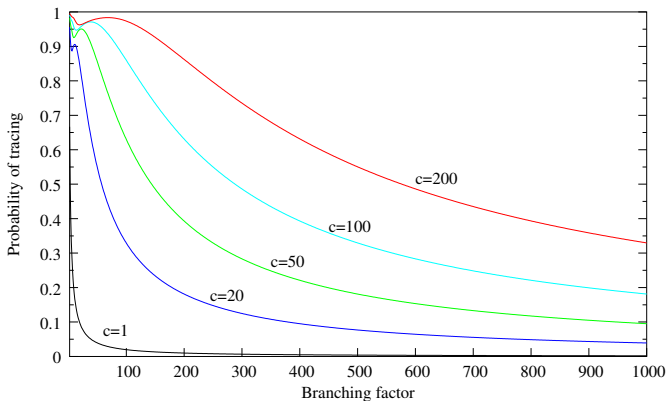


Molnar and Wagner Tree-Based Authentication

- Molnar and Wager suggested a tree-based technique to reduce the complexity from $O(n)$ to $O(\log n)$.
- The set of tags is recursively divided into **subgroups**. Each subgroup has its own key.



Analysis of the Resistance of Molnar and Wagner's Scheme

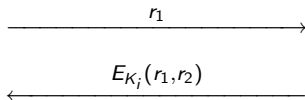


$$(n = 2^{20})$$

Verifier (Id_i, K_i)

database	
Id_1	K_1
Id_2	K_2
\vdots	\vdots
Id_i	K_i
\vdots	\vdots
Id_n	K_n

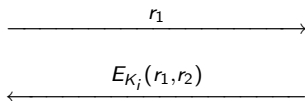
Prover (Id_i, K_i)



Verifier (Id_i, K_i)

Prover (Id_i, K_i)

database	
Id_1	K_1
Id_2	K_2
\vdots	\vdots
Id_i	K_i
\vdots	\vdots
Id_n	K_n



- Can the complexity be moved from the **on-line** phase to an **off-line** phase (precomputation)?

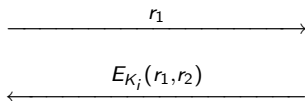
Pre-comp: 0 Storage: 0 On-line comp: n

Pre-comp: n Storage: n On-line comp: 0

Verifier (Id_i, K_i)

Prover (Id_i, K_i)

database	
Id_1	K_1
Id_2	K_2
\vdots	\vdots
Id_i	K_i
\vdots	\vdots
Id_n	K_n



- Can the complexity be moved from the **on-line** phase to an **off-line** phase (precomputation)?
 - Pre-comp: 0 Storage: 0 On-line comp: n
 - Pre-comp: n Storage: n On-line comp: 0
- The response must seem **random** from an adversary viewpoint, but be **predictable** by the verifier.

Verifier (s_i^1)

Prover (s_i^k)

→ (empty query)

← $r_i^k := G(s_i^k)$ $s_i^{k+1} = H(s_i^k)$

- Verifier and prover share a common value s_i^1
- G and H are **hash functions** (one-way, output “random”)

Computations Needed to Identify one Tag

- Receiving r_i^k , the reader computes from the initial secrets s_i^1 the hash chains until it finds r_i^k or until it reaches a given maximum limit m on the chain length.

$$\begin{array}{ccccccc} s_1^1 & \rightarrow & r_1^1 & r_1^2 & \dots & \dots & r_1^{m-1} & r_1^m \\ s_2^1 & \rightarrow & r_2^1 & r_2^2 & \dots & \dots & r_2^{m-1} & r_2^m \\ \vdots & \rightarrow & \dots & \dots & \dots & \dots & \dots & \vdots \\ s_i^1 & \rightarrow & \dots & \dots & \boxed{r_i^k = G(H^{k-1}(s_i^1))} & \dots & \dots & r_i^m \\ \vdots & \rightarrow & \dots & \dots & \dots & \dots & \dots & \vdots \\ s_n^1 & \rightarrow & r_n^1 & r_n^2 & \dots & \dots & r_n^{m-1} & r_n^m \end{array}$$

- The complexity in terms of **hash operation** is $2mn$.
- Storing the whole table is **not practicable**.

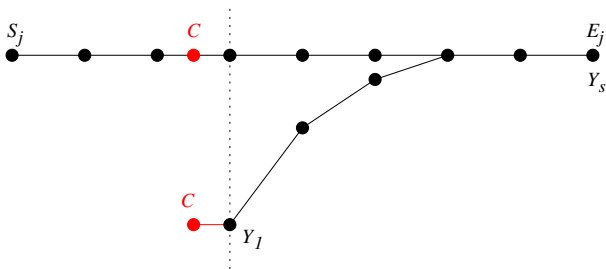
- The **general idea** of time-memory trade-offs is to reduce the time complexity of a given problem by adding memory (or the inverse).
- **Hellman's** trade-off (1980): $T = N^2/M^2$ e.g. $T = M = N^{2/3}$.
- Famous practical example: Windows **password cracking** (EPFL, 2003).
- Case study example: Inverting a **one-way function** (exhaustive search in order to find a preimage).

- Invert $S : A \rightarrow B$.
- Define $R : B \rightarrow A$ an arbitrary (reduction) function.
- Define $f : A \rightarrow A$ s.t. $f = R \circ S$.
- Chains are generated from arbitrary values in A .

$$\begin{array}{lclclclclclcl} S_1 & = & X_{1,1} & \xrightarrow{f} & X_{1,2} & \xrightarrow{f} & \dots & \xrightarrow{f} & X_{1,t} & = & E_1 \\ S_2 & = & X_{2,1} & \xrightarrow{f} & X_{2,2} & \xrightarrow{f} & \dots & \xrightarrow{f} & X_{2,t} & = & E_2 \\ & & \vdots & & & & & & & & \vdots \\ S_m & = & X_{m,1} & \xrightarrow{f} & X_{m,2} & \xrightarrow{f} & \dots & \xrightarrow{f} & X_{m,t} & = & E_m \end{array}$$

- Only the first and the last element of each chain is stored.
- The generated values should cover the set A .

- Given one output $C \in B$, we compute $Y_1 := R(C)$ and generate a chain starting at Y_1 : $Y_1 \xrightarrow{f} Y_2 \xrightarrow{f} Y_3 \xrightarrow{f} \dots Y_s$



- In our case, the function to invert is:

$$(i, k) \mapsto r_i^k = G(H^{k-1}(s_i^1)) \quad (1 \leq i \leq n, 1 \leq k \leq m).$$

- Comparison

Scheme	Time (millisecond)
Without Precomputation	16'000
With Precomputation (342 MB)	0.122
With Precomputation (1.25 GB)	0.002

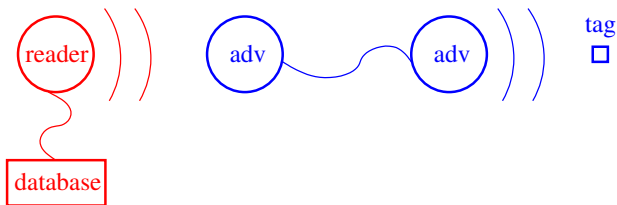
$$(n = 2^{20}, m = 2^{10})$$

- Constructive approach of Time-Memory Trade-Off

Other (Funny) Problems in Constrained Environments

Relay Attacks

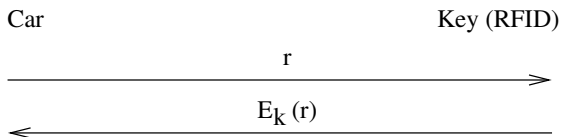
The reader believes that the tag is **within its electromagnetic field** while it is not the case (the attacker behaves as an **extension cord**).



Countermeasures consist in computing the **round trip time**. Issues are authentication of the packet and practicability.

- The main concepts of cryptography, i.e, confidentiality, integrity, and authentication, are treated **without any practical considerations**.
- If one of these properties is **theoretically** ensured, it remains ensured in practice whatever the layer we choose to implement the protocol.
- Privacy needs to be ensured at **each layer**.
- All efforts to prevent traceability in the application layer **may be useless** if no care is taken at the lower layers.

Attack of Bono *et al.* against the Digital Signature Transponder manufactured by Texas Instrument, used in automobile ignition key (there exist more than **130 millions** such keys).



Cipher (not public) uses **40 bit keys**: active attack in **less than 1 minute** (time-memory trade-offs)

- 1 Recovering the cryptographic key
- 2 Impersonating the car ignition key
- 3 Impersonating the SpeedPass payment card

Summary and Conclusion

- **Current main research topic:** (privacy-compliant) authentication protocols in constrained environments (RFID).
- **Focus in this talk:** how to design efficient challenge-response protocols (one way of research among others), two works presented (attack on Molnar-Wagner, construction using time-memory trade-offs)
- **Other (funny) problems introduced:** relay attacks, traceability thanks to the lower layers, weak building blocks (more funny problems: blocker tag, noisy tag, side-channel attacks, etc.)
- <http://lasecwww.epfl.ch/~gavoine/rfid>