

A Scalable and Provably Secure Hash-Based RFID Protocol

Gildas Avoine and Philippe Oechslin

EPFL, Lausanne, Switzerland



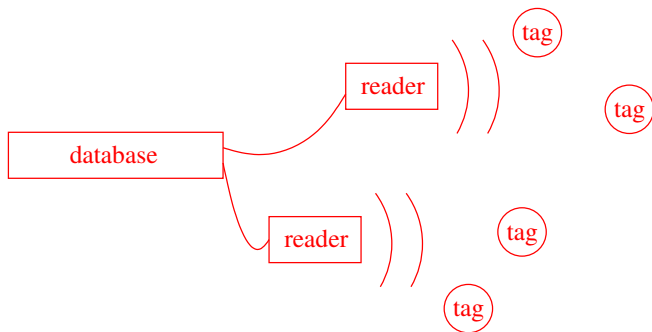
A Brief Introduction to the RFID Technology

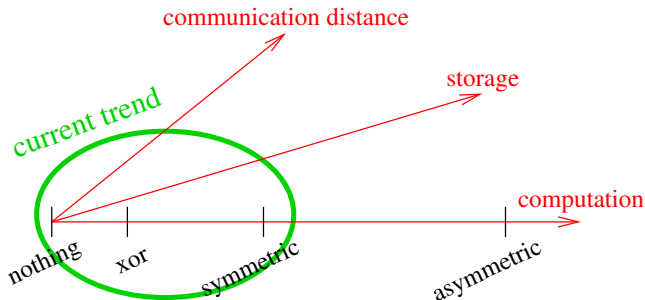
The Ohkubo-Suzuki-Kinoshita (OSK) Protocol

Time-Memory Trade-Off Techniques Applied to OSK

A Brief Introduction to the RFID Technology

Radio Frequency Identification: Identification of objects remotely by embedding in these objects tiny devices (**tags**) capable of transmitting data.





The boom which RFID technology is enjoying today relies essentially on the willingness to develop **small** and **cheap** RFID tags.

- Management of stocks
- Pets identification
- Localization of people
- Libraries
- Speed up the checkouts in the shops
- Recycling
- Anti-counterfeiting
- Sensor networks

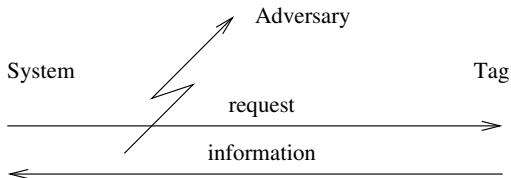
The main security issue of the RFID technology is the **traceability** of people according to the tags they carry.

Easier to trace people using the RFID technology than other technologies because tags cannot be **switched-off** and can be almost **invisible**. Moreover, it is easy to automatically analyze the **logs** of the readers and the trend is to increase the **communication distance**.

Traceability: *Given a set of readings between tags and readers, an adversary must not be able to find any relation between any readings of a same tag or set of tags.*

Forward traceability: *Given a set of readings between tags and readers and given the fact that **all information** stored in the involved tags has been **revealed** at time t , the adversary must not be able to find any relation between any readings of a same tag or set of tags that occurred at a time $t' \leq t$.*

An RFID protocol should be such that an **authorized party is able to identify** a tag while an **adversary should not be able to trace** it.



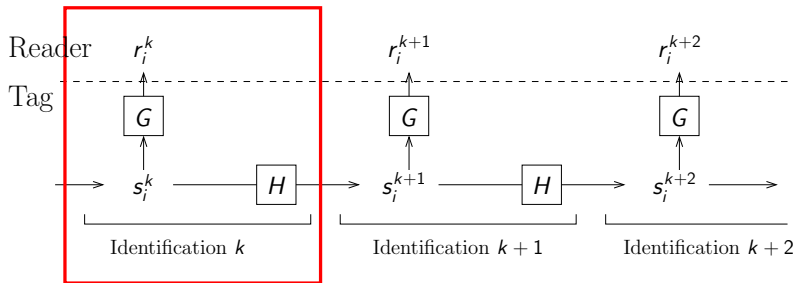
From this **information**, the system is able to **identify** the tag.

Adversary should neither be able to **identify** the tag nor to **track** it.

Information needs to be **randomized**.

The Ohkubo-Suzuki-Kinoshita (OSK) Protocol

- Each tag needs 2 **hash functions** G and H (in theory).
- Each tag needs an **EEPROM** capable of storing an identifier.
- The personalisation of a tag T_i consists in storing in its memory a random identifier s_i^1 , which is also recorded by the database of the system.
- Thus, the database initially contains the set $\{s_i^1 \mid 1 \leq i \leq n\}$.



From each n initial identifiers s_i^1 , the system computes 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 **average** complexity in terms of **hash operation** is mn (if the tag is owned by the system) because two hash operations are carried out $mn/2$ times.

Example: in a library, we consider 2^{20} tagged books. The length of the hash chains is 2^{10} . We assume that the system can carry out 2^{24} hash operations per second.

The storage complexity is **16 megabytes** and the average computation complexity is 2^{30} i.e. **1 minute**.

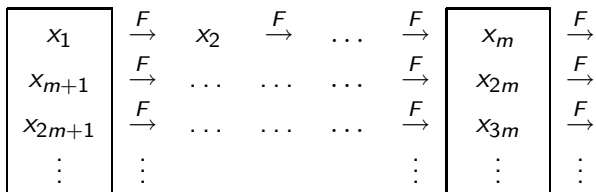
Time-Memory Trade-Off Techniques Applied to OSK

The **general idea** of time-memory trade-offs is to reduce the time complexity by adding memory (or the inverse).

Example: exhaustive search on a one-way function $F: X \rightarrow X$ in order to find preimages. $N := |X|$

Extreme cases:

Storage: 0	Pre-computation: 0	On-line computation: N
Storage: N	Pre-computation: N	On-line computation: 0



Only the **first** and the **last** element of each chain is stored.

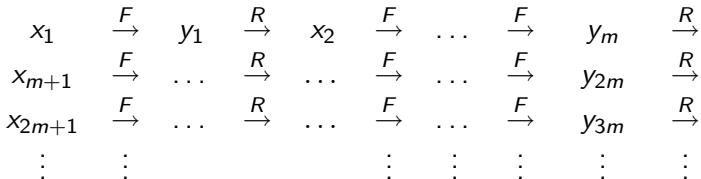
Given one output x_j of F that we need to invert, we generate a chain starting at x_j : $x_j \xrightarrow{F} x_{j+1} \xrightarrow{F} x_{j+2} \xrightarrow{F} \dots$

Each time we compute a new element, we check whether or not it is the **end of a chain** in the table.

We can then regenerate the complete chain and find x_{j-1} .

In practice, **inputs** and **outputs** of F are not in the same set X but $F: X \rightarrow Y$.

We need a **reduction** function $R: Y \rightarrow X$ that generates an arbitrary input of F from one of its outputs.



The computation time is $T_{to} = N^2\gamma/M^2$ where γ is a small factor depending on the probability of success and the particular type of trade-off being used [Oech03].

In our case the function F is

$$F : (i, k) \mapsto r_i^k = G(H^{k-1}(s_i^1))$$

where $1 \leq i \leq n$ and $1 \leq k \leq m$.

F is more complex than usual trade-offs because it is not a simple hash function: i and k are arbitrary results from R and we need $m/2 + 1$ hash operations to compute $F(i, k)$.

We need to store the n values s_i^1 in order to compute F .

We can optimize by storing intermediate elements of the chain, sacrificing so memory but **reducing the average complexity** of F .

We need an arbitrary reduction function R ,

$$R : r_i^k \mapsto (i', k')$$

where $1 \leq i' \leq n, 1 \leq k' \leq m$. For example, we take

$$R(r) = (1 + (r \bmod n), 1 + (\lfloor \frac{r}{n} \rfloor \bmod m)).$$

Since the brute force method needs n units of memory to store the n initial identifiers s_i^1 , the amount of memory used by the trade-off is measured as a multiple c of this required memory.

$2(|n| + |m|)$ bits are required to store one chain while $|s|$ bits per chain of identifiers are needed in the brute force approach. The **conversion factor** is $\mu = |s|/(2|n| + 2|m|)$.

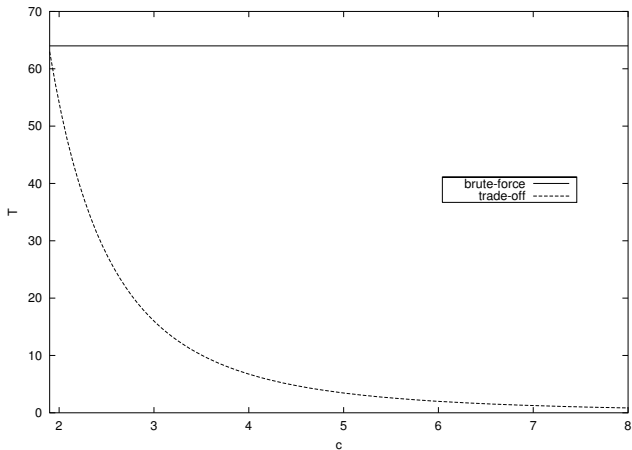
We have :

$$T_{to} \approx \left(\frac{3m}{2c}\right)^3 \frac{\gamma}{\mu^2} \quad \text{and} \quad \frac{T_{bf}}{T_{to}} \approx \left(\frac{2c}{3}\right)^3 \frac{\mu^2 n}{m^2 \gamma}.$$

- Number of tags: $n = 2^{20}$
- Hash-chain length: $m = 2^{10}$
- Output length of F : $s = 128$
- Storing 2 chains required less memory than storing one s : $\mu \leq 2$
- Single back-end with 1 GB of memory: $c = 64$
- We choose a success rate of 99.9%: $\gamma = 8$

$$T_{to} \approx \left(\frac{3m}{2c}\right)^3 \frac{\gamma}{\mu^2} \quad \text{i.e. 0.0016 seconds}$$

$$T_{precalc} \approx \frac{nm^2}{2} \quad \text{i.e. 10 hours}$$



More information on Security and Privacy in RFID Systems

<http://lasecwww.epfl.ch/~gavoine/rfid/>