# Cryptanalysis of T-function-Based Hash Functions

*ICISC 2006*

Frédéric Muller[1] and Thomas Peyrin[2]

[1] HSBC France

[2] France Télécom R&D

December 1, 2006

# Outline

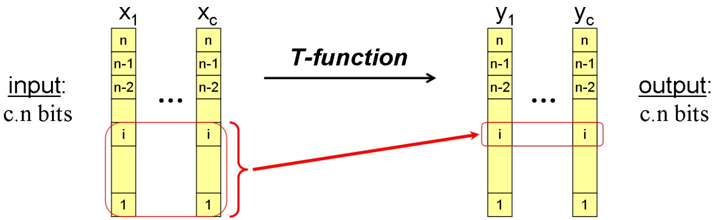
# Outline

# Cryptographic Hash Functions Nowadays

- Building an efficient and secure compression function is not easy !
- Dedicated design hash functions: are MD-x and SHA-x secure (Wang *et al.*) ?
- Block cipher-based hash functions: no well accepted candidate.
- Can we use other primitives to build good compression functions ? => maybe T-functions !

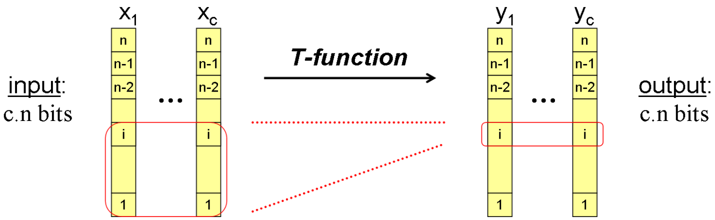
# Reminder of T-functions

- Introduced in 2002 by Klimov and Shamir.
- T-function (Triangular Function): mapping from *c* *n*-bit words to *c* *n*-bit words, where the *i*-th bit of any output word depends only on the *i* LSBs of the input words.



# Reminder of T-functions

- Introduced in 2002 by Klimov and Shamir.
- T-function (Triangular Function): mapping from $c$ $n$-bit words to $c$ $n$-bit words, where the $i$-th bit of any output word depends only on the $i$ LSBs of the input words.



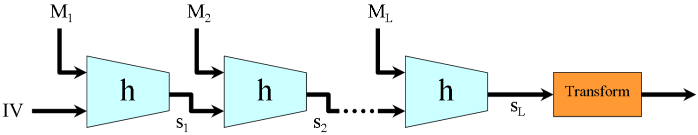
# Properties of T-functions

- Very fast primitives: most usual software operations are T-functions ($+$, $\times$, bitwise logic operations).
- Composition of T-functions is a T-function.
- Can be used to build very efficient and easy to analyze stream ciphers or MDS nonlinear mappings for block ciphers.
- Is it possible to use T-functions for hash functions design ?

# Outline

# First Considerations

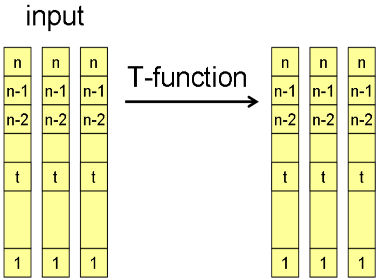
- Reminder of Merkle-Damgård iteration:




- We do not directly build a hash function with T-functions but a compression function *h*.
- We need to find a way to reduce the output length of the T-function to get a compression function.

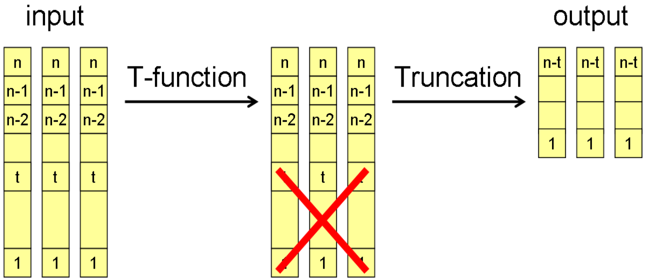
# The Truncation Method

- The MSB of the output of a T-function reveal little information about the input (triangular structure).
- Truncate the output and keep only the MSB of the output words.



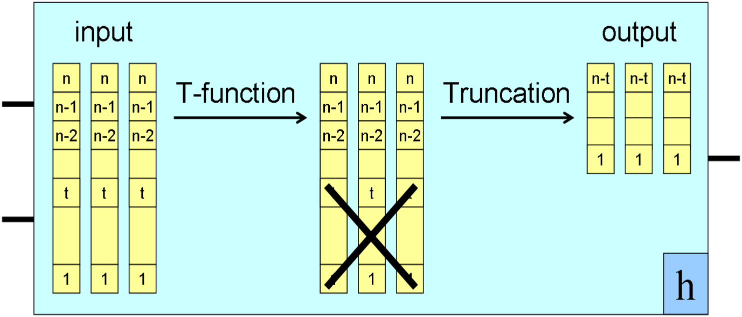
# The Truncation Method

- The MSB of the output of a T-function reveal little information about the input (triangular structure).
- Truncate the output and keep only the MSB of the output words.



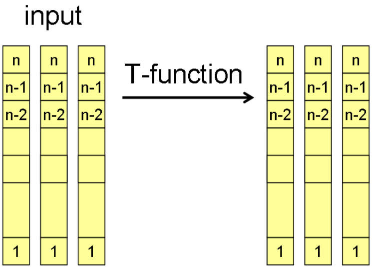
# The Truncation Method

- The MSB of the output of a T-function reveal little information about the input (triangular structure).
- Truncate the output and keep only the MSB of the output words.



# The Reduction Method

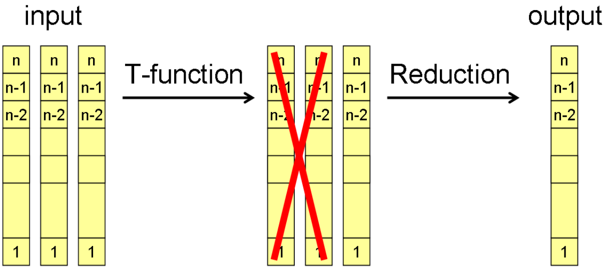
- Another natural solution: keep only some of the output words.




- Example: the MySQL dedicated hash function.

# The Reduction Method

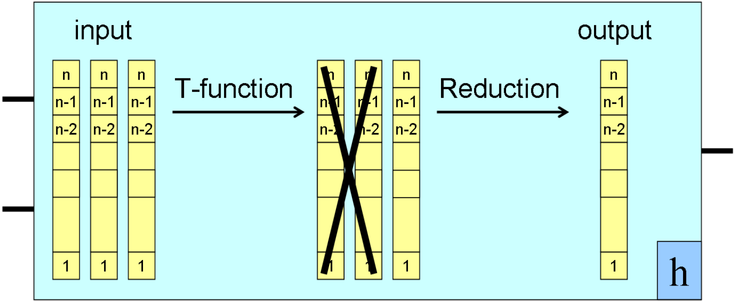
- Another natural solution: keep only some of the output words.




- Example: the MySQL dedicated hash function.

# The Reduction Method

- Another natural solution: keep only some of the output words.



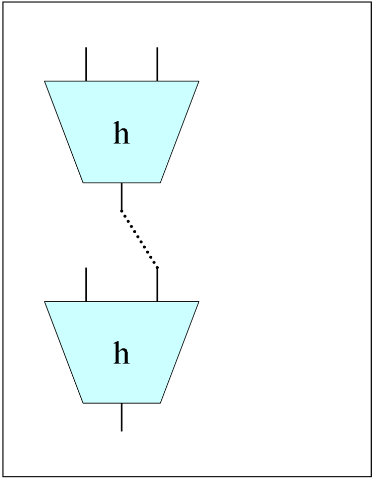

- Example: the MySQL dedicated hash function.

# Outline

# Preimages and Pseudo-preimages

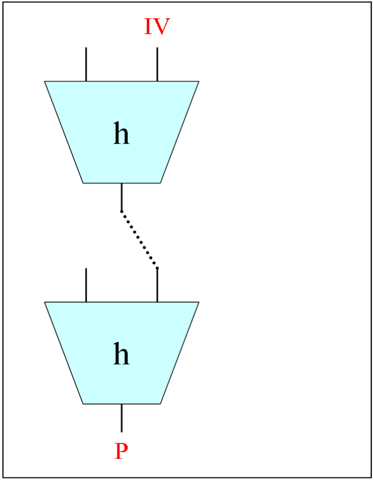
- Preimage: given $P$ and $IV$, find a message $M$ such that $h(IV, M) = P$.
- Pseudo-preimage: given $P$, find a message $M$ and a value $IV$ such that $h(IV, M) = P$.
- A pseudo-preimage attack on a compression function $h$ can be transformed into a preimage attack against the whole hash function $H$: meet-in-the-middle attack.
- This does not apply to (pseudo)-collision resistance.

# The Meet in the Middle Attack



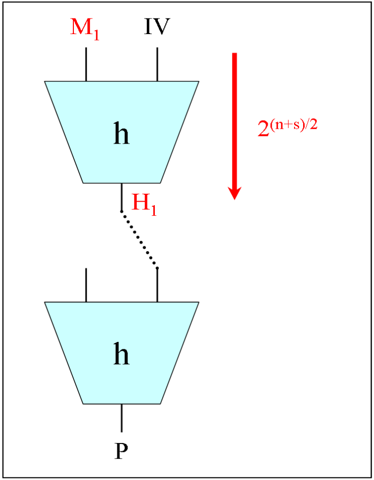

- We are given $P$ and $IV$.
- Compute $2^{\frac{n+s}{2}}$ values $H_1 = h(IV, M_1)$.
- Compute $2^{\frac{n-s}{2}}$ pseudo-preimages $(H_2, M_2)$ of $P$.
- Find a match between $H_1$ and $H_2$.
- If a pseudo-preimage can be found in $\Theta(2^s)$ for $h$, a preimage for $H$ can be found in $\Theta(2^{1+\frac{n+s}{2}})$.

# The Meet in the Middle Attack



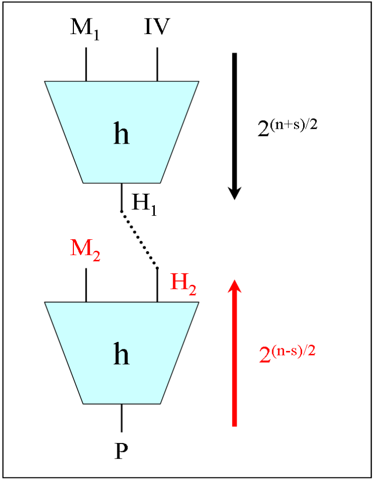

- We are given $P$ and $IV$.
- Compute $2^{\frac{n+s}{2}}$ values $H_1 = h(IV, M_1)$.
- Compute $2^{\frac{n-s}{2}}$ pseudo-preimages $(H_2, M_2)$ of $P$.
- Find a match between $H_1$ and $H_2$.
- If a pseudo-preimage can be found in $\Theta(2^s)$ for $h$, a preimage for $H$ can be found in $\Theta(2^{1+\frac{n+s}{2}})$.

# The Meet in the Middle Attack



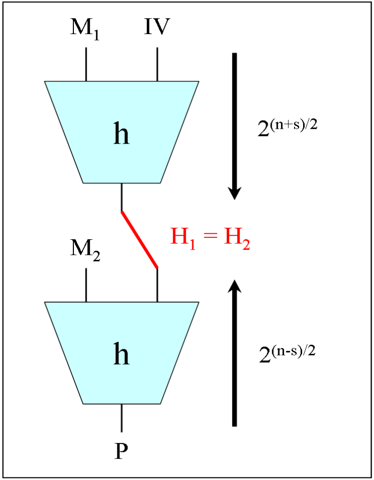

- We are given $P$ and $IV$.
- Compute $2^{\frac{n+s}{2}}$ values $H_1 = h(IV, M_1)$.
- Compute $2^{\frac{n-s}{2}}$ pseudo-preimages $(H_2, M_2)$ of $P$.
- Find a match between $H_1$ and $H_2$.
- If a pseudo-preimage can be found in $\Theta(2^s)$ for $h$, a preimage for $H$ can be found in $\Theta(2^{1+\frac{n+s}{2}})$.

# The Meet in the Middle Attack



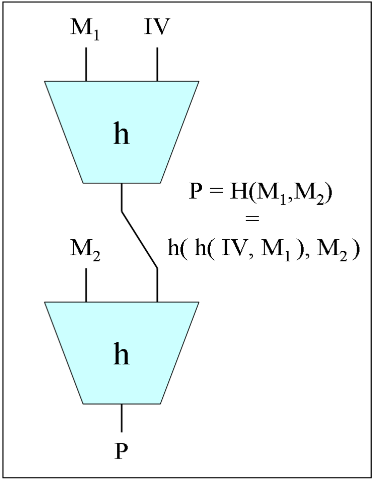

- We are given $P$ and $IV$.
- Compute $2^{\frac{n+s}{2}}$ values $H_1 = h(IV, M_1)$.
- Compute $2^{\frac{n-s}{2}}$ pseudo-preimages $(H_2, M_2)$ of $P$.
- Find a match between $H_1$ and $H_2$.
- If a pseudo-preimage can be found in $\Theta(2^s)$ for $h$, a preimage for $H$ can be found in $\Theta(2^{1+\frac{n+s}{2}})$.

# The Meet in the Middle Attack




- We are given $P$ and $IV$.
- Compute $2^{\frac{n+s}{2}}$ values $H_1 = h(IV, M_1)$.
- Compute $2^{\frac{n-s}{2}}$ pseudo-preimages $(H_2, M_2)$ of $P$.
- Find a match between $H_1$ and $H_2$.
- If a pseudo-preimage can be found in $\Theta(2^s)$ for $h$, a preimage for $H$ can be found in $\Theta(2^{1+\frac{n+s}{2}})$.

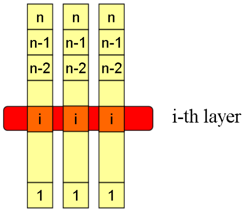
# The Meet in the Middle Attack




- We are given $P$ and $IV$.
- Compute $2^{\frac{n+s}{2}}$ values $H_1 = h(IV, M_1)$.
- Compute $2^{\frac{n-s}{2}}$ pseudo-preimages $(H_2, M_2)$ of $P$.
- Find a match between $H_1$ and $H_2$.
- If a pseudo-preimage can be found in $\Theta(2^s)$ for $h$, a preimage for $H$ can be found in $\Theta(2^{1+\frac{n+s}{2}})$.

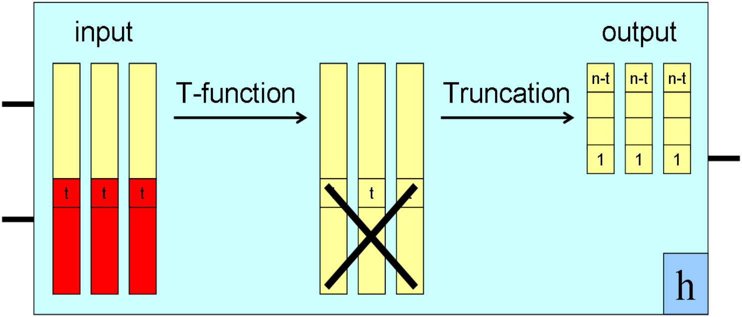
# Attacking The Compression Function

- We have to be able to compute pseudo-preimages or pseudo-collisions for the compression function.
- Idea: attack layer by layer.
- Works for the truncation or the reduction method.
- Analysis of complexity given in the paper.



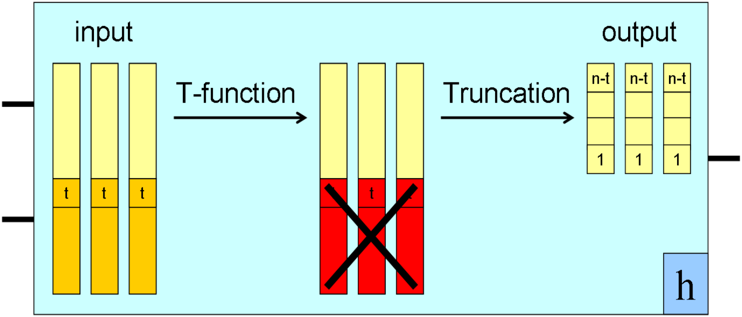
# Attacking The Truncation Method (1)

- Fix an arbitrary value for the $t$ LSB of the input blocks (the truncated bits).



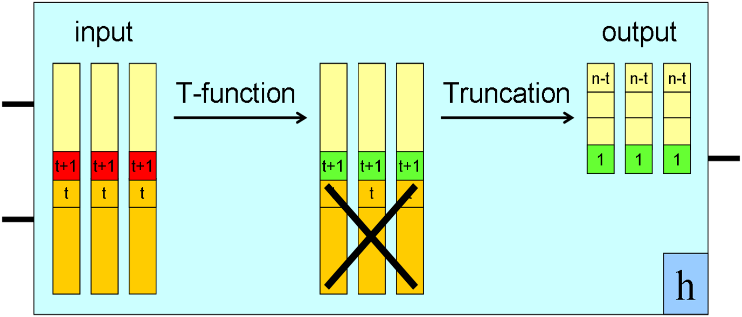
# Attacking The Truncation Method (2)

- The *t* LSB of the output blocks are now determined (but there is no constraint on these values).



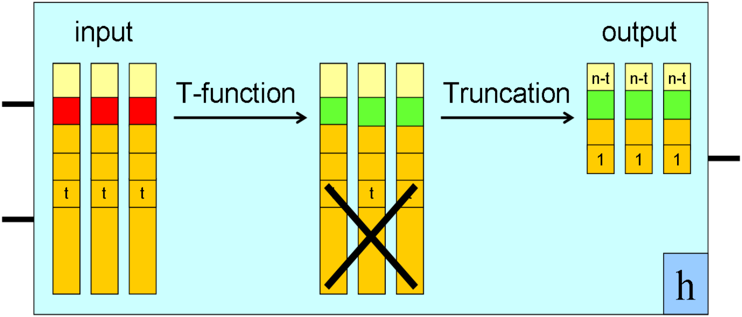
# Attacking The Truncation Method (3)

- Find a value for the $(t+1)$-th layer that remains consistant for a candidate preimage and continue to the next layer.



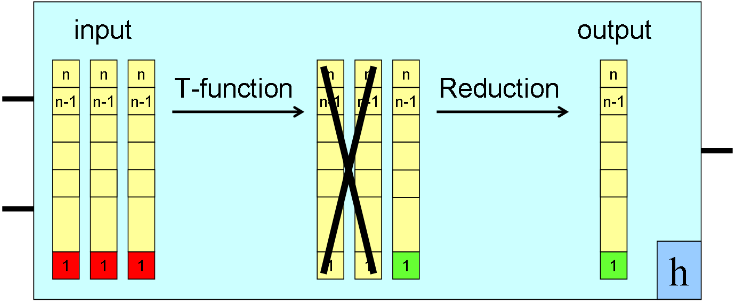
# Attacking The Truncation Method (4)

- If at some point no good value exists, then restart the algorithm (or backtrack).



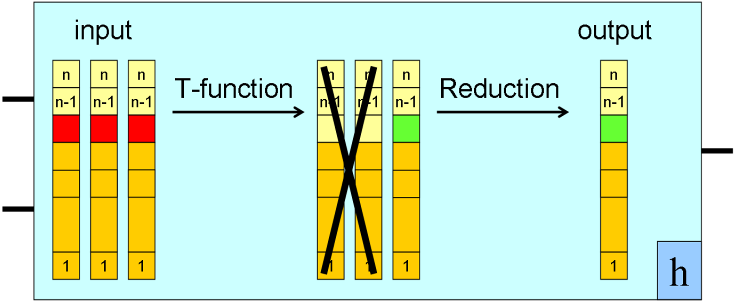
# Attacking The Reduction Method (1)

- Find a value for the first layer that remains consistant for a candidate preimage and continue to the next layer.



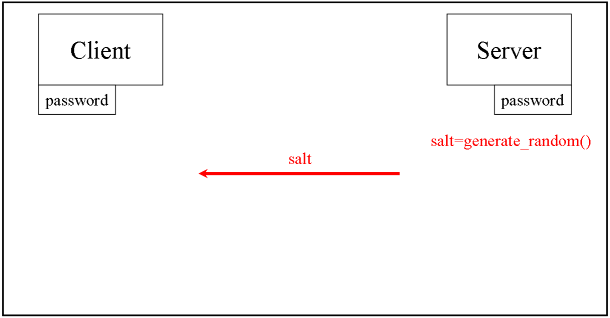
# Attacking The Reduction Method (2)

- If at some point no good value exists, then restart the algorithm (or backtrack).



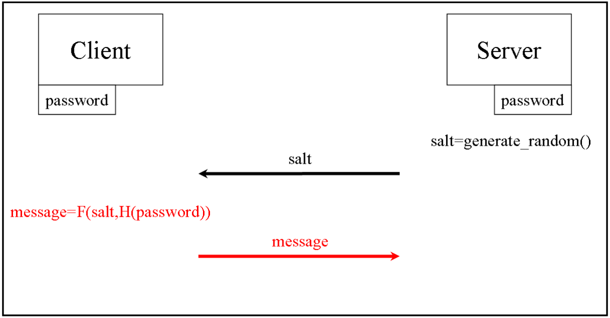
# Outline

# The Authentication Protocol



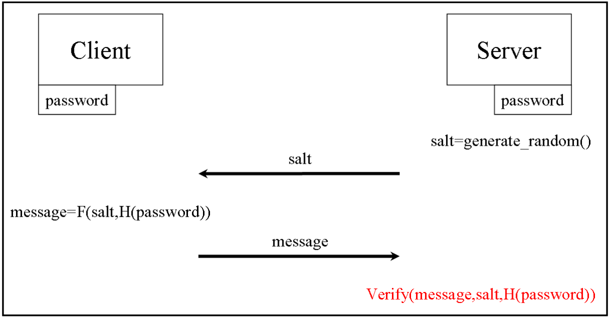

- H is an iterative hash function and F is a PRNG whose seed is derived from the salt and H(password).

# The Authentication Protocol




- H is an iterative hash function and F is a PRNG whose seed is derived from the salt and H(password).

# The Authentication Protocol




- H is an iterative hash function and F is a PRNG whose seed is derived from the salt and H(password).

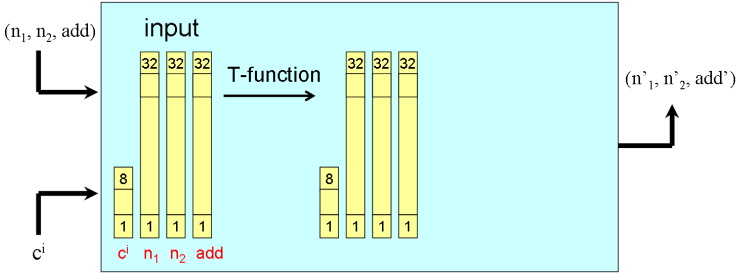
# The Different MySQL Authentication Versions

- Until version 3.23 (september 2003), MySQL used a dedicated hash function H and a dedicated scrambling function F.
- In 2002, an attack against F was found, but no attack against H is known yet.
- New versions use SHA-1 as basic component (see our paper) but old ones continue to be implemented due to compatibility issues.

# Specification of the Old Dedicated Hash Function

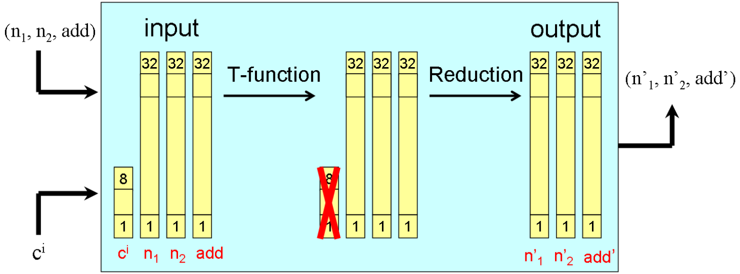

- At iteration $i$, the compression function $h$ updates a 96-bit chaining variable $s^i = n_1^i||n_2^i||add^i$ with one password byte $c^i$.

# Specification of the Old Dedicated Hash Function




- At iteration $i$, the compression function $h$ updates a 96-bit chaining variable $s^i = n_1^i || n_2^i || add^i$ with one password byte $c^i$.

# Specification of the Old Dedicated Hash Function




- At iteration $i$, the compression function $h$ updates a 96-bit chaining variable $s^i = n_1^i || n_2^i || add^i$ with one password byte $c^i$.

# Specification of the Old Dedicated Compression Function

- At the end of the whole iteration process, we only keep the 30 LSB of $n_1$ and $n_2$ for a 60-bit output.
- The compression function $h$ is a T-function (reduction method):

$$\begin{aligned} n_1^i &= n_1^{i-1} \oplus (((n_1^{i-1} \wedge 63) + add^{i-1}) \cdot c^i + (n_1^{i-1} \ll 8)) \\ n_2^i &= (n_2^{i-1} \ll 8) \oplus n_1^{i-1} \\ add^i &= add^{i-1} + c^i \end{aligned}$$

# Outline

# The Cryptanalysis Scenario

- The attacker listens to communication between the client and the server and learns pairs (salt,message).
- He can impersonate the legitimate client by breaking the scrambling function.
- But to know the actual value of the password, he needs to break the preimage resistance of $H$.
- The compression function $h$ is a T-function following the reduction method, we apply our generic attack and then we find a preimage for $H$ using the meet-in-the-middle attack.

# Analysis and Results

- We implemented this attack !
- We restricted ourselves to keyboard reachable characters (about 100).
- The password (or a "equivalent" one) can be found in less than a second on a standard PC.
- Example: 'MySQL123' and 'RGp0mA23' hash to the same output.

# Outline

# Conclusions

- "Natural" hash functions based on T-functions are weak due to generic attacks.
- Be careful with old-version MySQL authentication: we can retrieve the user's password in less than a second.
- Is there a way to build secure non-natural T-function-based compression functions ?
- See our article for a deeper analysis of the (old) scrambling function.