Rainbow Tables

ENEE 457

How are Passwords Stored?

- Option 1: Store all passwords in a table in the clear.
  - Problem: If Server is compromised then all passwords are leaked.
- Option 2: Store only the hash values in a table in the clear.
  - If Server is compromised, hard to recover password values given hash values.

Background

- Cryptographic hash function H.
- Given $H(x)$ it is hard to find x' such that $H(x') = H(x)$.
- How hard is it?
  - Assume “brute force” is the best attack
  - Try all possible passwords x' and check whether $H(x') = H(x)$.
- How many possible passwords are there?
  - Assume a dictionary of size N.
  - E.g. if passwords are 6 characters (case sensitive letters, numerals, special characters) then $N \approx 95^6$.

Simple Time-Memory Trade Offs

- Can run brute force attack each time to invert the hash:
  - $O(N)$ time, $O(1)$ memory
- Can precompute the entire truth table, use a lookup each time to invert the hash:
  - $O(1)$ time (depending on data structure), $O(N)$ memory.

A Cryptanalytic Time-Memory Trade Off

Construction of Table (pre-processing):

- Choose m starting points:

$$SP_1 \coloneqq X_{1,0}, \dots, SP_m \coloneqq X_{m,0}$$

- Compute $X_{i,j} = f\left(X_{i,j-1}\right) = R\left(H\left(X_{i,j-1}\right)\right)$
- Reduction function R is a mapping from the range of the hash to the dictionary D.
  - E.g. take first six characters of hash output.
- $EP_i = f^t(SP_i)$

$$\begin{array}{c}
SP_1 = X_{10} \xrightarrow{f} X_{11} \xrightarrow{f} X_{12} \xrightarrow{f} \cdots \xrightarrow{f} X_{1t} = EP_1 \\
SP_2 = X_{20} \xrightarrow{f} X_{21} \xrightarrow{f} X_{22} \xrightarrow{f} \cdots \xrightarrow{f} X_{2t} = EP_2 \\
\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
SP_m = X_{m0} \xrightarrow{f} X_{m1} \xrightarrow{f} X_{m2} \xrightarrow{f} \cdots \xrightarrow{f} X_{mt} = EP_m
\end{array}$$

- Save the pairs $\{EP_i, SP_i\}_{1 \le i \le m}$

A Cryptanalytic Time-Memory Trade Off

Looking up a hash inverse:

- Given $h^* = H(m)$:
  - Apply R to obtain $Y_1 = R(h^*) = f(m)$
  - Check if Y_1 is an endpoint in the table.
  - If yes ($Y_1 = EP_i$), recompute from SP_i to find pre-image.
  - Otherwise, compute $Y_2 = f(Y_1)$ and repeat.
  - Do this until reaching $Y_t = f^t(Y_1)$.

Success Probability?

- Heuristic argument—need m, t to each be approx. $\sqrt{N}$ to have good success probability.

Problem:

- Not all intermediate values in chains will be unique.
- “Collisions” $\rightarrow$ “Merges” of chains
  - So after a collision, the chain is useless.

Theorem (Hellman '80)

The success probability $P(S)$ is at least

$$P(S) \geq \left(\frac{1}{N}\right) \sum_{i=1}^{m} \sum_{j=0}^{t-1} \left[\frac{N - it}{N}\right]^{j+1}$$

Proof of Theorem

Let A be the set of distinct entries in the set of m chains of length t. Then $P(S) = E[|A|]/N$.

Let $I_{i,j}$ be the indicator variable set to 1 if position (i, j) is a "new" value (when filling in the table row-by-row starting from $i = 1$) and set to 0 otherwise.

$$E[|A|] = \sum_{i=1}^{m} \sum_{j=0}^{t-1} E[I_{i,j}] = \sum_{i=1}^{m} \sum_{j=0}^{t-1} P\left(I_{i,j} = 1\right)$$

$$P(I_{i,j} = 1) \geq P\left(I_{i,0} = 1 \wedge I_{i,1} = 1 \wedge \cdots \wedge I_{i,j} = 1\right)$$
$$= P\left(I_{i,0} = 1\right) \cdot P\left(I_{i,1} = 1 \mid I_{i,0} = 1 \right) \cdots P\left(I_{i,j} = 1 \mid I_{i,0} = 1 \cdots I_{i,j-1} = 1 \right)$$
$$= \frac{N - |A_i|}{N} \cdot \frac{N - |A_i| - 1}{N} \cdots \frac{N - |A_i| - j}{N}$$
$$\geq \left[\frac{N - it}{N}\right]^{j+1}$$

Where A_i is the set of distinct elements at the moment we reach the i-th row. Clearly, $|A_i| \leq it$.

Parameter Settings

- Set $m, t \ := N^{\frac{1}{3}}$
- $P(S) \geq 1/N^{1/3}$

Storing ℓ independent tables

- Increase success probability from $P(S)$ to
$$1 - \left(1 - P(S)\right)^{\ell}.$$

Optimal Parameters

- Set $m, t, \ell := N^{\frac{1}{3}}$
- Require storage of size $N^{2/3}$, each lookup requires $N^{2/3}$ computations.
- For our example before,
  - Brute force search $95^6 \approx 7 \times 10^{11}$.
  - Using Hellman's method $95^4 \approx 8 \times 10^7$
  - 10^{-6} second per hash
  - ≈ 194 hours (8 days) to invert one hash value vs. 80 seconds.

Rivest's Modification ('82)

- Distinguished endpoints
  - E.g. the first ten bits are zero
- When given a hash value to invert, can generate a chain of keys until we find a distinguished point and only then look it up in the memory.
- Greatly reduces the number of memory lookups
- Allow for loop detection
- Merges can be easily detected since two merging chains will have the same endpoint.

Rainbow Tables

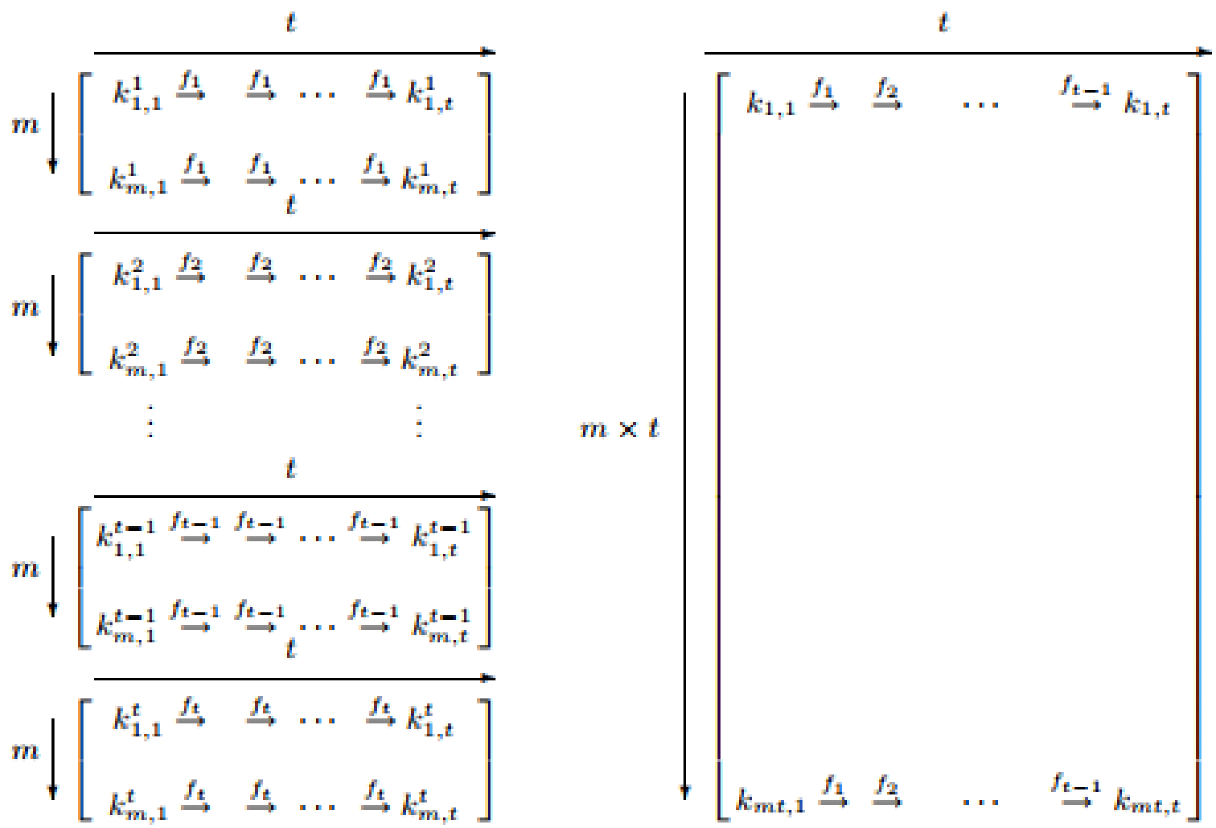
- Introduced by Philippe Oechslin in ‘03.
  - “Making a Faster Cryptanalytic Time-Memory Trade-Off”
- Modifies Hellman’s method:
  - Chains use a successive reduction function for each point in the chain—”rainbow”.
  - Start with reduction function 1 and end with reduction function $t-1$.
  - For chains of length t, if a collision occurs, the chance of it being a merge is only $\frac{1}{t}$ (collision must occur in same column).
- ****Collisions do not necessarily imply merges****

Additional Benefits of Rainbow Tables

- The number of table look-ups is reduced by a factor of t compared to Hellman's method.
- Merges of chains result in identical endpoints, so they are detectable and can be eliminated from table.
- No loops.
- Rainbow chains have constant length (as opposed to distinguished points).

Success Probability

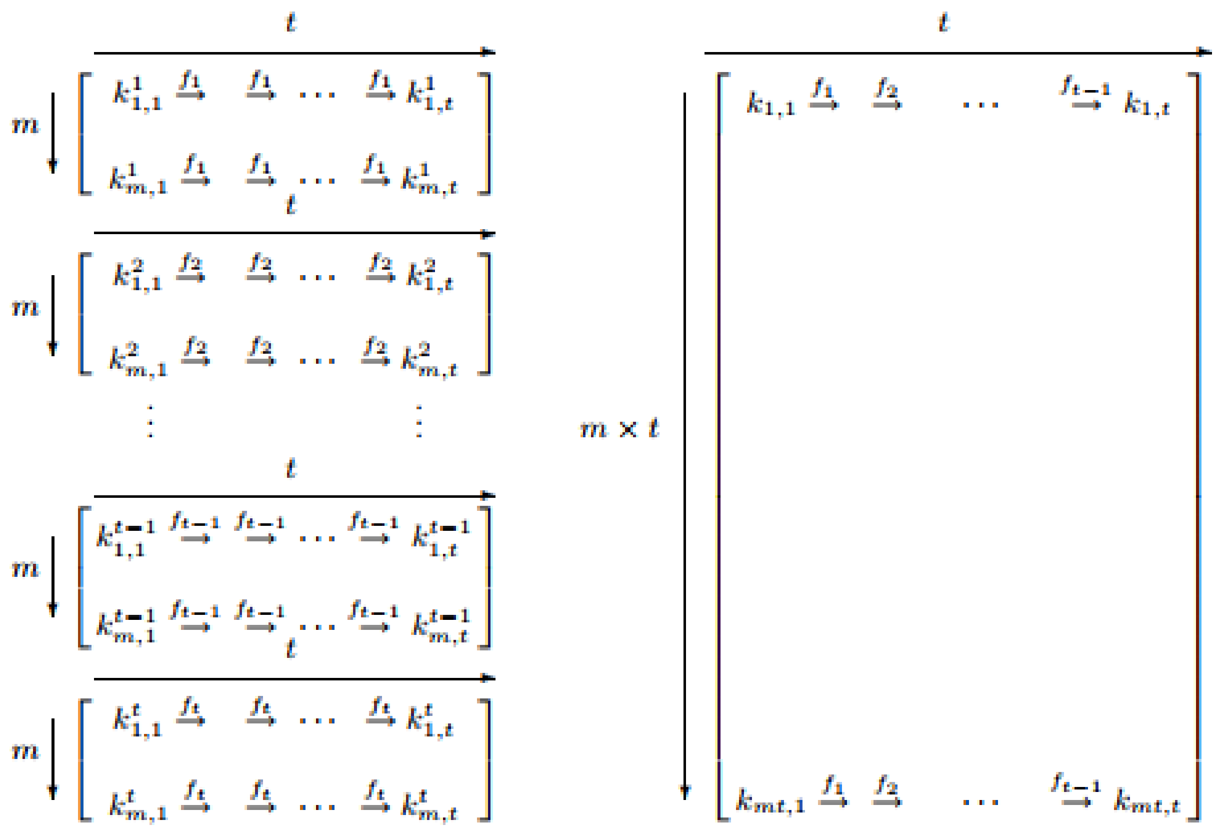
Success probability of t classical tables of size $m \times t$ is approximately equal to that of a single rainbow table of size $mt \times t$.



Lookup Time

Lookup requires t^2 calculations in classical table

Can be done with $1 + 2 + \cdots t = \frac{t(t-1)}{2}$ calculations in Rainbow table



Countermeasure Against Rainbow Tables

- Rainbow Table takes advantage of the fact that N is fairly small.
- Countermeasure: Store $H(password||salt)$
  - $salt$ is public and can be stored along with the hash
- Attacker would need to precompute a table for every possible $salt$ value.
- E.g. 128-bit salt would require 2^{128} tables.