Permutation-Based Hash Chains with Application to Password Hashing

Charlotte Lefevre, Bart Mennink
Radboud University (The Netherlands)

FSE 2025

17 March 2025

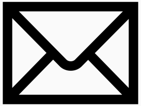
Introduction

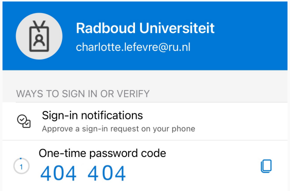
Second-Factor Authentication

- Many systems rely on password-based authentication

Second-Factor Authentication

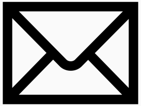
- Many systems rely on password-based authentication
- Second-factor authentication mitigates the weaknesses of passwords

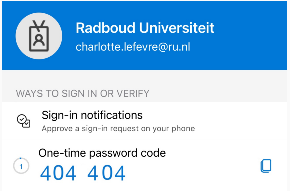





Second-Factor Authentication

- Many systems rely on password-based authentication
- Second-factor authentication mitigates the weaknesses of passwords





- Some use time-based one-time password schemes

TOTP: Time-Based One-Time Password Algorithm [MMPR11] (RFC 6238)

- Client and Server share a secret key κ

TOTP: Time-Based One-Time Password Algorithm [MMPR11] (RFC 6238)

- Client and Server share a secret key κ
- The password is generated by applying

$$\mathrm{HOTP}(\kappa, \mathrm{ctr}) = \mathrm{Truncate}\,(\mathrm{HMAC}(\kappa, \mathrm{ctr}))$$

where:

  - HMAC may be $\mathrm{HMAC\text{-}SHA\text{-}256}$ or $\mathrm{HMAC\text{-}SHA\text{-}512}$
  - ctr is a counter based on the current time (typically changes every 30s)
  - $\mathrm{Truncate}$ truncates the output to a 6-digit number

TOTP: Time-Based One-Time Password Algorithm [MMPR11] (RFC 6238)

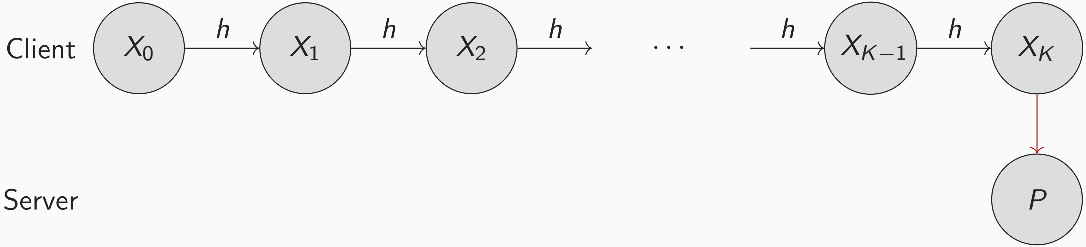
- Client and Server share a secret key κ
- The password is generated by applying

$$\mathrm{HOTP}(\kappa, \mathrm{ctr}) = \mathrm{Truncate}\,(\mathrm{HMAC}(\kappa, \mathrm{ctr}))$$

  where:

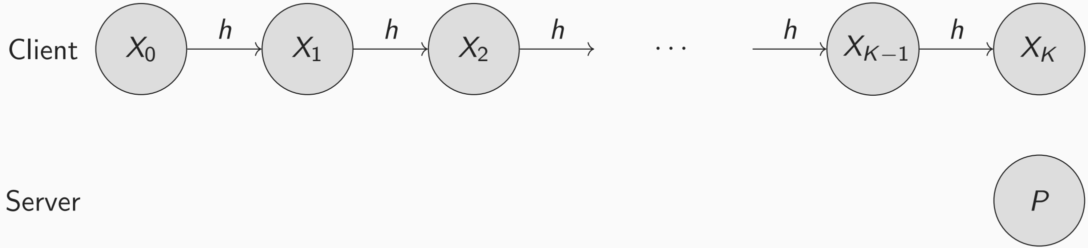
  - HMAC may be $\mathrm{HMAC\text{-}SHA\text{-}256}$ or $\mathrm{HMAC\text{-}SHA\text{-}512}$
  - ctr is a counter based on the current time (typically changes every 30s)
  - $\mathrm{Truncate}$ truncates the output to a 6-digit number
- **Downside:** Server must securely store κ

Hash Chains and S/Key [Lam81, Hal95] (RFC 1760)



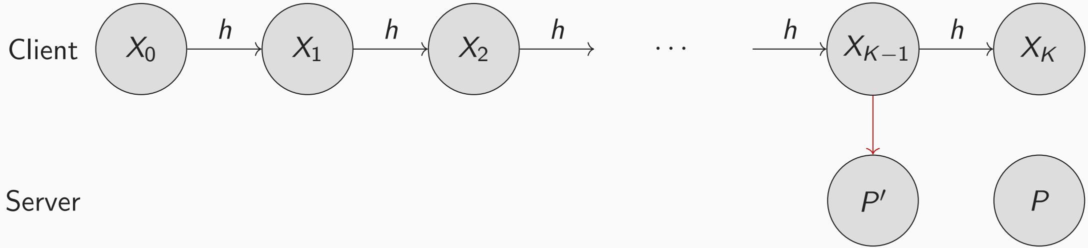
- Client generates X_0, sends securely $P = h^K(X_0)$ to Server

Hash Chains and S/Key [Lam81, Hal95] (RFC 1760)



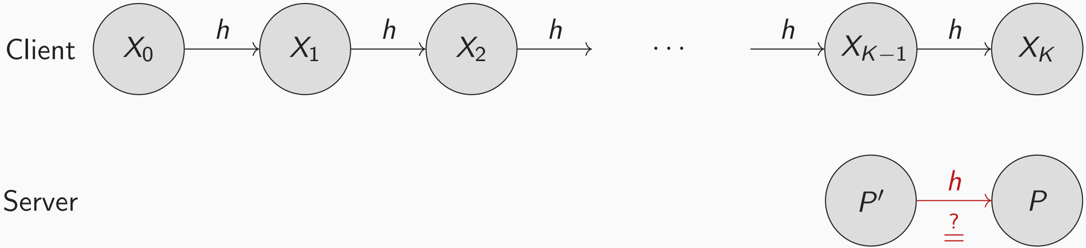
- Client generates X_0, sends securely $P = h^K(X_0)$ to Server

Hash Chains and S/Key [Lam81, Hal95] (RFC 1760)



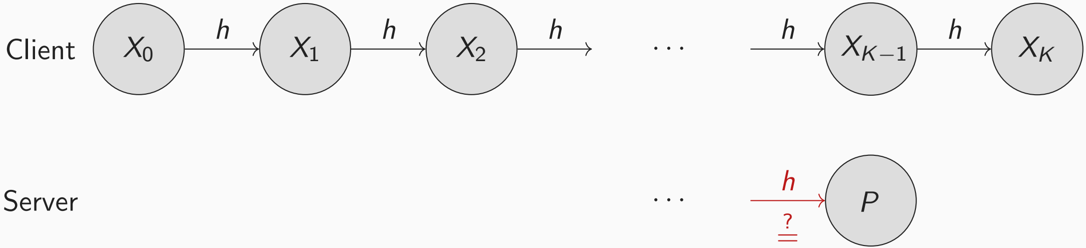
- Client generates X_0, sends securely $P = h^K(X_0)$ to Server
- At authentication round number k, Client sends $P' = h^{K-k}(X_0)$ to Server

Hash Chains and S/Key [Lam81, Hal95] (RFC 1760)



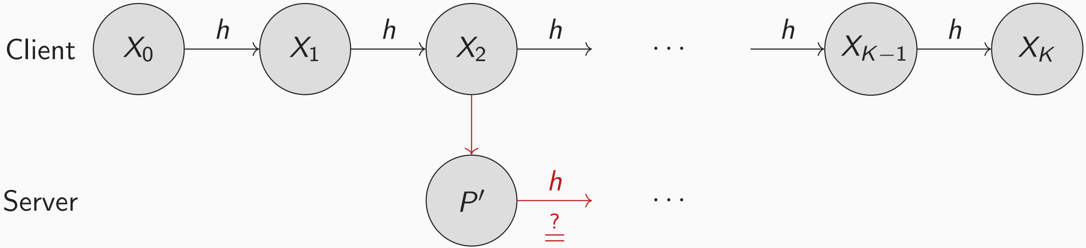
- Client generates X_0, sends securely $P = h^K(X_0)$ to Server
- At authentication round number k, Client sends $P' = h^{K-k}(X_0)$ to Server
- Server checks whether $h(P') = P$

Hash Chains and S/Key [Lam81, Hal95] (RFC 1760)



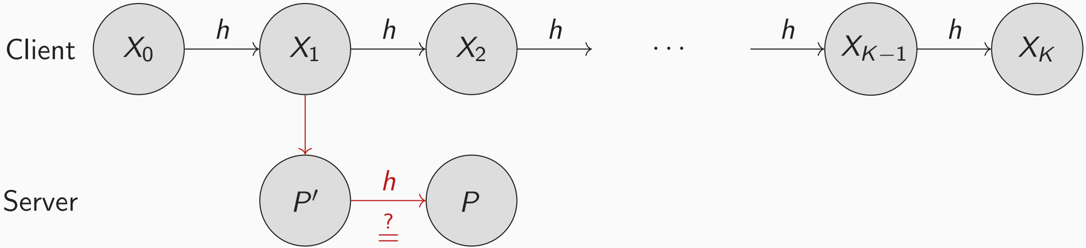
- Client generates X_0, sends securely $P = h^K(X_0)$ to Server
- At authentication round number k, Client sends $P' = h^{K-k}(X_0)$ to Server
- Server checks whether $h(P') = P$

Hash Chains and S/Key [Lam81, Hal95] (RFC 1760)



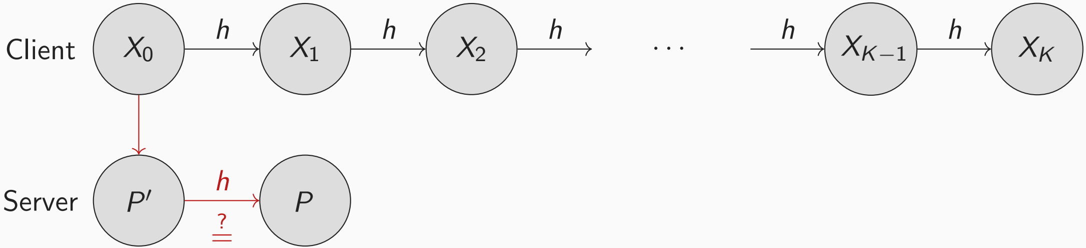
- Client generates X_0, sends securely $P = h^K(X_0)$ to Server
- At authentication round number k, Client sends $P' = h^{K-k}(X_0)$ to Server
- Server checks whether $h(P') = P$

Hash Chains and S/Key [Lam81, Hal95] (RFC 1760)



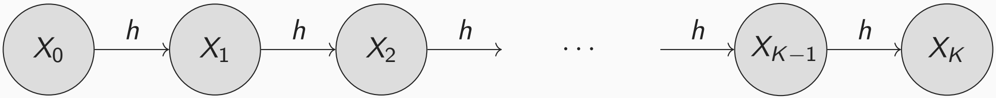
- Client generates X_0, sends securely $P = h^K(X_0)$ to Server
- At authentication round number k, Client sends $P' = h^{K-k}(X_0)$ to Server
- Server checks whether $h(P') = P$

Hash Chains and S/Key [Lam81, Hal95] (RFC 1760)



- Client generates X_0, sends securely $P = h^K(X_0)$ to Server
- At authentication round number k, Client sends $P' = h^{K-k}(X_0)$ to Server
- Server checks whether $h(P') = P$

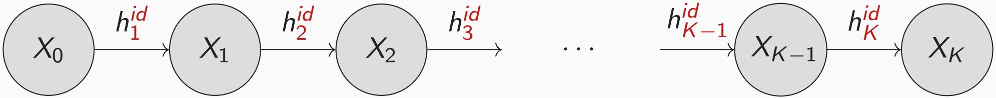
Hash Chains and S/Key (cont'd)



Weaknesses

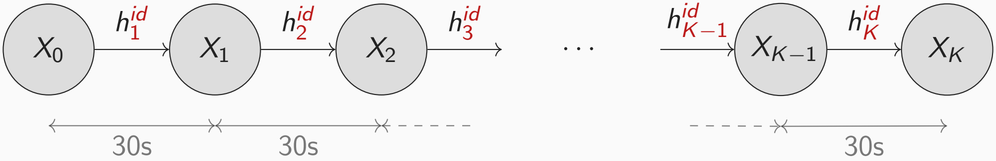
- Not time-based: increases attack window if logins are scarce
- Iterating a hash function weakens its security $\Longrightarrow$ security degradation by factor K

T/Key [KMB17]



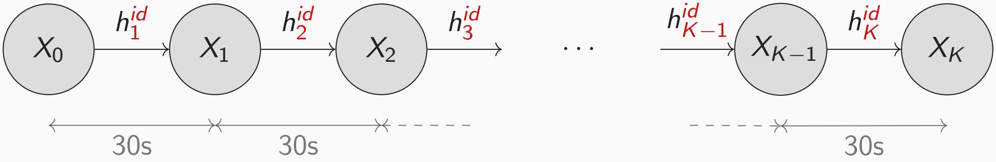
- Domain separation and salt incorporation: $h_k^{id}(\cdot) := h(\langle \mathrm{ctr}_k \rangle_t \parallel id \parallel \cdot)$:
  - $\langle \mathrm{ctr}_k \rangle_t$ is timestamp encoded over t bits
  - id is s-bit random salt
- X_0 is n-bit uniformly random string

T/Key [KMB17]



- Domain separation and salt incorporation: $h_k^{id}(\cdot) := h(\langle \mathrm{ctr}_k \rangle_t \parallel id \parallel \cdot)$:
  - $\langle \mathrm{ctr}_k \rangle_t$ is timestamp encoded over t bits
  - id is s-bit random salt
- X_0 is n-bit uniformly random string
- Every point on the chain is valid for a limited amount of time

T/Key [KMB17]



- Domain separation and salt incorporation: $h_k^{id}(\cdot) := h(\langle \mathrm{ctr}_k \rangle_t \parallel id \parallel \cdot)$:
    - $\langle \mathrm{ctr}_k \rangle_t$ is timestamp encoded over t bits
    - id is s-bit random salt
- X_0 is n-bit uniformly random string
- Every point on the chain is valid for a limited amount of time
- Suggestion by the designers:

$s = 80$ $t = 32$ $K = 2^{21}$ $n = 130$ 30s time frames

Security of T/Key

Assuming that h is a random oracle, T/Key is secure up to bound [KMB17]

$$\frac{2(q+K)}{2^n}$$

where q denotes the number of queries to h

Security of T/Key

Assuming that h is a random oracle, T/Key is secure up to bound [KMB17]

$$\frac{2(q+K)}{2^n}$$

where q denotes the number of queries to h

Our Contribution: refined and improved security of hash chains

Assuming that h is a random oracle, T/Key is secure up to bound [KMB17]

$$\frac{2(q+K)}{2^n}$$

where q denotes the number of queries to h

Our Contribution: refined and improved security of hash chains

1. Capture the time-based release pattern of T/Key in a security model

Security of T/Key

Assuming that h is a random oracle, T/Key is secure up to bound [KMB17]

$$\frac{2(q+K)}{2^n}$$

where q denotes the number of queries to h

Our Contribution: refined and improved security of hash chains

1. Capture the time-based release pattern of T/Key in a security model
2. Refined security proof with a random oracle

Assuming that h is a random oracle, T/Key is secure up to bound [KMB17]

$$\frac{2(q+K)}{2^n}$$

where q denotes the number of queries to h

Our Contribution: refined and improved security of hash chains

1. Capture the time-based release pattern of T/Key in a security model
2. Refined security proof with a random oracle
3. Dedicated security analysis with the sponge construction

Assuming that h is a random oracle, T/Key is secure up to bound [KMB17]

$$\frac{2(q+K)}{2^n}$$

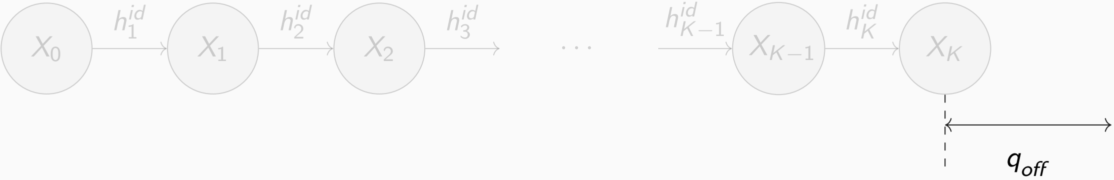
where q denotes the number of queries to h

Our Contribution: refined and improved security of hash chains

1. Capture the time-based release pattern of T/Key in a security model
2. Refined security proof with a random oracle
3. Dedicated security analysis with the sponge construction
4. Dedicated security analysis with a truncated permutation

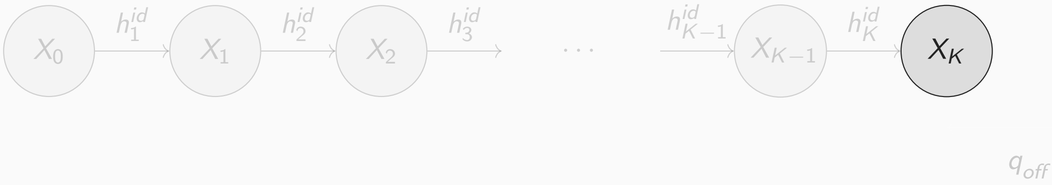
Security Model

Security Model



- h is a hash function construction based on an ideal primitive $\mathcal{P}$
- Offline phase: $\mathcal{A}$ makes q_{off} queries to $\mathcal{P}$

Security Model

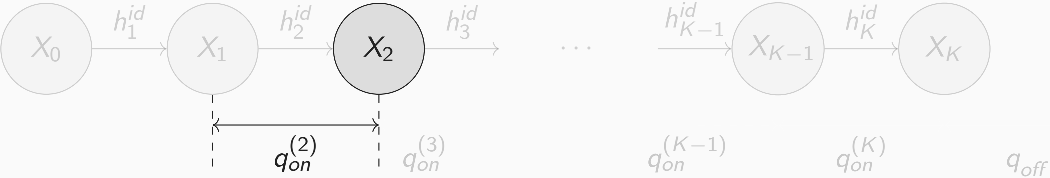


- h is a hash function construction based on an ideal primitive $\mathcal{P}$
- Offline phase: $\mathcal{A}$ makes q_{off} queries to $\mathcal{P}$
- Online phase:

- h is a hash function construction based on an ideal primitive $\mathcal{P}$
- Offline phase: $\mathcal{A}$ makes q_{off} queries to $\mathcal{P}$
- Online phase: $\mathcal{A}$ makes $q_{on}^{(k)}$ queries to $\mathcal{P}$ at each step

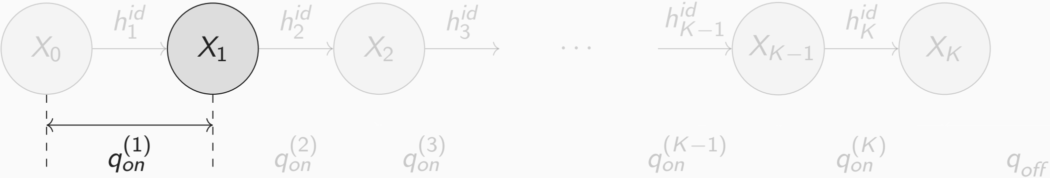
Security Model

- h is a hash function construction based on an ideal primitive $\mathcal{P}$
- Offline phase: $\mathcal{A}$ makes q_{off} queries to $\mathcal{P}$
- Online phase: $\mathcal{A}$ makes $q_{on}^{(k)}$ queries to $\mathcal{P}$ at each step



- h is a hash function construction based on an ideal primitive $\mathcal{P}$
- Offline phase: $\mathcal{A}$ makes q_{off} queries to $\mathcal{P}$
- Online phase: $\mathcal{A}$ makes $q_{on}^{(k)}$ queries to $\mathcal{P}$ at each step

Security Model



- h is a hash function construction based on an ideal primitive $\mathcal{P}$
- Offline phase: $\mathcal{A}$ makes q_{off} queries to $\mathcal{P}$
- Online phase: $\mathcal{A}$ makes $q_{on}^{(k)}$ queries to $\mathcal{P}$ at each step

Security Model

- h is a hash function construction based on an ideal primitive $\mathcal{P}$
- Offline phase: $\mathcal{A}$ makes q_{off} queries to $\mathcal{P}$
- Online phase: $\mathcal{A}$ makes $q_{on}^{(k)}$ queries to $\mathcal{P}$ at each step
- $\mathcal{A}$ wins if they invert any of the X_k within the livespan of that X_k

Security Model

- h is a hash function construction based on an ideal primitive $\mathcal{P}$
- Offline phase: $\mathcal{A}$ makes q_{off} queries to $\mathcal{P}$
- Online phase: $\mathcal{A}$ makes $q_{on}^{(k)}$ queries to $\mathcal{P}$ at each step
- $\mathcal{A}$ wins if they invert any of the X_k within the livespan of that X_k
- Security advantage is denoted by $\mathbf{Adv}_h^{\mathrm{T/Key}}(q_{off}, q_{on}, M)$, where
  - $q_{on} = \sum_k q_{on}^{(k)}$ (we assume $q_{on} \ll 2^{100}$)
  - M denotes the number of users

- Bound from T/Key designers:

$$\mathbf{Adv}_{\mathcal{RO}}^{\mathrm{T/Key}}(q_{\mathit{off}}, q_{\mathit{on}}, 1) = \mathcal{O}\left(\frac{q_{\mathit{off}} + q_{\mathit{on}}}{2^n}\right)$$

Security of T/Key in the New Model with a Random Oracle

- Bound from T/Key designers:

$$\mathbf{Adv}^{\mathrm{T/Key}}_{\mathcal{RO}}(q_{off}, q_{on}, 1) = \mathcal{O}\left(\frac{q_{off} + q_{on}}{2^n}\right)$$

- We prove (up to logarithmic factors):

$$\mathbf{Adv}^{\mathrm{T/Key}}_{\mathcal{RO}}(q_{off}, q_{on}, M) = \mathcal{O}\left(\frac{M}{2^s} \cdot \frac{q_{off}}{2^n} + \max\left(\frac{M}{2^s}; 1\right) \cdot \frac{q_{on}}{2^n}\right)$$

Security of T/Key in the New Model with a Random Oracle

- Bound from T/Key designers:

$$\textbf{Adv}^{\mathrm{T/Key}}_{\mathcal{RO}}(q_{\mathit{off}}, q_{\mathit{on}}, 1) = \mathcal{O}\left(\frac{q_{\mathit{off}} + q_{\mathit{on}}}{2^n}\right)$$

- We prove (up to logarithmic factors):

$$\textbf{Adv}^{\mathrm{T/Key}}_{\mathcal{RO}}(q_{\mathit{off}}, q_{\mathit{on}}, M) = \mathcal{O}\left(\frac{M}{2^s} \cdot \frac{q_{\mathit{off}}}{2^n} + \max\left(\frac{M}{2^s}; 1\right) \cdot \frac{q_{\mathit{on}}}{2^n}\right)$$

- Assuming $q_{\mathit{off}} \ll 2^{128}$, $q_{\mathit{on}} \ll 2^{100}$, and $M \ll 2^{52}$, password size can be reduced from 130 to 100

Security of T/Key with a Hash Function Construction

- Let $\mathcal{H}$ be a hash function construction:

$$\mathbf{Adv}_{\mathcal{H}}^{\mathrm{T/Key}}(q_{\mathit{off}}, q_{\mathit{on}}, 1) \leq \mathbf{Adv}_{\mathcal{RO}}^{\mathrm{T/Key}}(q_{\mathit{off}}, q_{\mathit{on}}, 1) + \mathbf{Adv}_{\mathcal{H}}^{\mathrm{iff}}(q_{\mathit{off}} + q_{\mathit{on}})$$

where $\mathbf{Adv}_{\mathcal{H}}^{\mathrm{iff}}$ denotes the indifferentiability advantage of $\mathcal{H}$

Security of T/Key with a Hash Function Construction

- Let $\mathcal{H}$ be a hash function construction:

$$\mathbf{Adv}_{\mathcal{H}}^{\mathrm{T/Key}}(q_{\mathit{off}}, q_{\mathit{on}}, 1) \leq \mathbf{Adv}_{\mathcal{RO}}^{\mathrm{T/Key}}(q_{\mathit{off}}, q_{\mathit{on}}, 1) + \mathbf{Adv}_{\mathcal{H}}^{\mathrm{iff}}(q_{\mathit{off}} + q_{\mathit{on}})$$

  where $\mathbf{Adv}_{\mathcal{H}}^{\mathrm{iff}}$ denotes the indifferentiability advantage of $\mathcal{H}$

- But indifferentiability is overkill:
  - Online/offline separation lost with indifferentiability
  - The actual security property is a complex variant of preimage resistance

- Let $\mathcal{H}$ be a hash function construction:

$$\mathbf{Adv}_{\mathcal{H}}^{\text{T/Key}}(q_{\mathit{off}}, q_{\mathit{on}}, 1) \leq \mathbf{Adv}_{\mathcal{RO}}^{\text{T/Key}}(q_{\mathit{off}}, q_{\mathit{on}}, 1) + \mathbf{Adv}_{\mathcal{H}}^{\text{iff}}(q_{\mathit{off}} + q_{\mathit{on}})$$

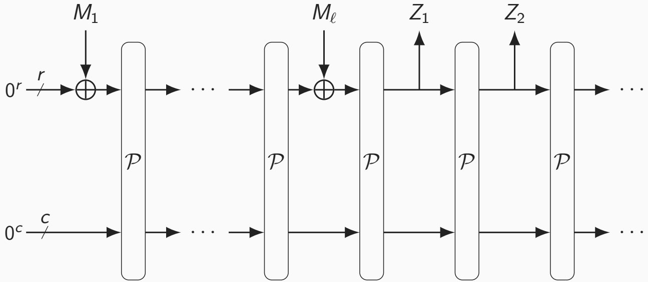
where $\mathbf{Adv}_{\mathcal{H}}^{\text{iff}}$ denotes the indifferentiability advantage of $\mathcal{H}$

- But indifferentiability is overkill:
  - Online/offline separation lost with indifferentiability
  - The actual security property is a complex variant of preimage resistance

A dedicated analysis will likely give a better bound

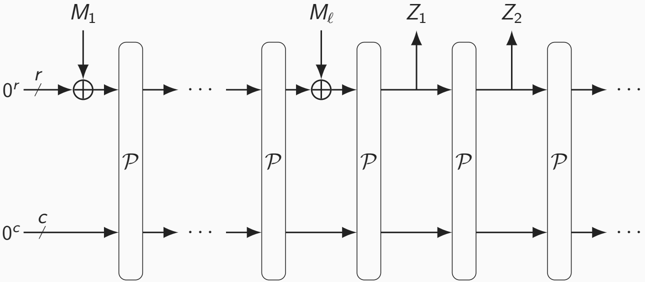
T/Key with the Sponge Construction

The Sponge Construction [BDPV07]



- Permutation $\mathcal{P}$ of size $b = r + c$
- $M_1 \| \cdots \| M_\ell$ is the message padded into r-bit blocks

The Sponge Construction [BDPV07]



- Permutation $\mathcal{P}$ of size $b = r + c$
- $M_1 \| \cdots \| M_\ell$ is the message padded into r-bit blocks
- The sponge construction has a tight indifferentiability bound [BDPV08]:

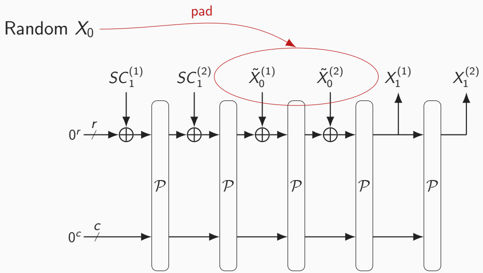
$$\mathbf{Adv}^{\mathrm{iff}}_{\mathrm{Sponge}}(q) \leq \frac{q(q+1)}{2^c}$$

T/Key with the Sponge Construction

Random X_0

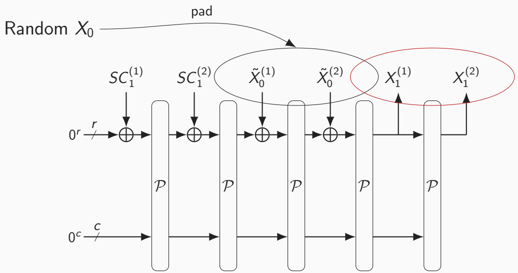
- Assume that $s + t = 2r$ and $n + 1 = 2r$

T/Key with the Sponge Construction



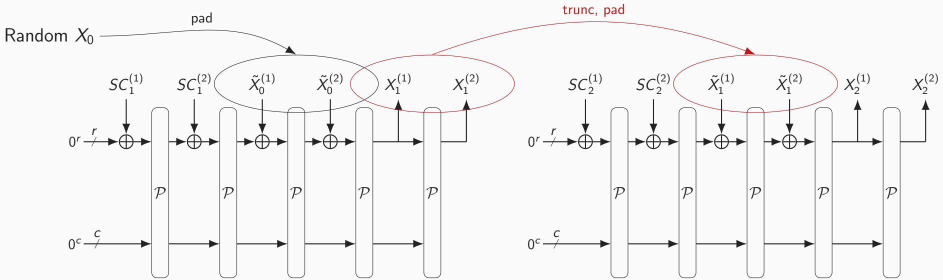
- Assume that $s + t = 2r$ and $n + 1 = 2r$
- $SC_k^{(i)}$ represents the salt and counter blocks
- $\tilde{X}_k^{(i)}$ represents a password block after padding

T/Key with the Sponge Construction



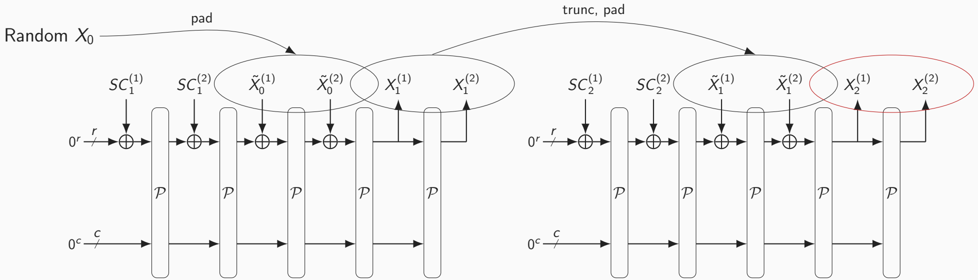
- Assume that $s + t = 2r$ and $n + 1 = 2r$
- $SC_k^{(i)}$ represents the salt and counter blocks
- $\tilde{X}_k^{(i)}$ represents a password block after padding

T/Key with the Sponge Construction



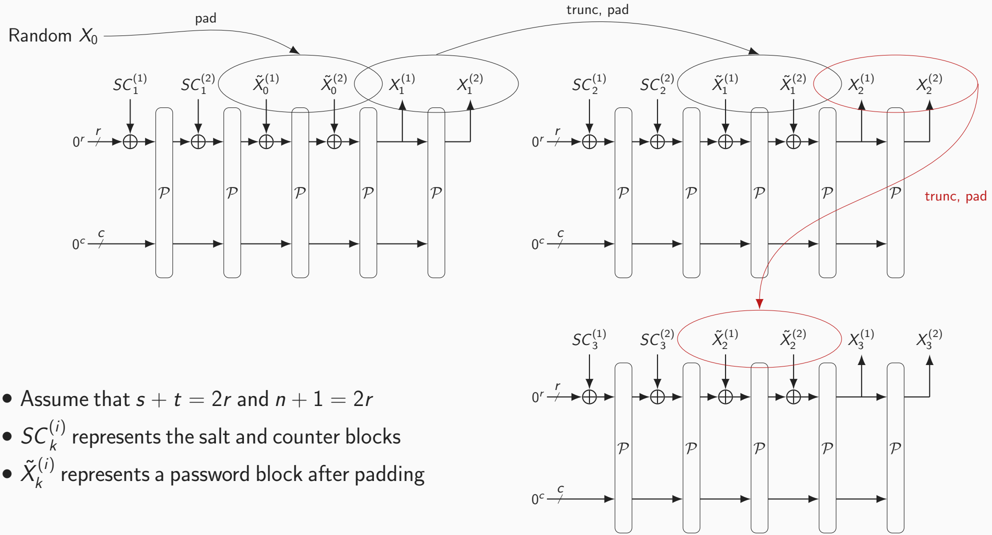
- Assume that $s + t = 2r$ and $n + 1 = 2r$
- $SC_k^{(i)}$ represents the salt and counter blocks
- $\tilde{X}_k^{(i)}$ represents a password block after padding

T/Key with the Sponge Construction



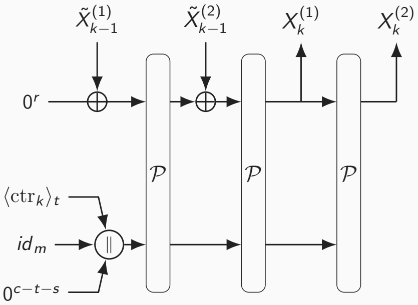
- Assume that $s + t = 2r$ and $n + 1 = 2r$
- $SC_k^{(i)}$ represents the salt and counter blocks
- $\tilde{X}_k^{(i)}$ represents a password block after padding

T/Key with the Sponge Construction



- Assume that $s + t = 2r$ and $n + 1 = 2r$
- $SC_k^{(i)}$ represents the salt and counter blocks
- $\tilde{X}_k^{(i)}$ represents a password block after padding

T/Key with the Sponge Construction: Optimization



- Salt and counter are part of the initial state
- Requires that $s + t \leq c$
- Our security bound holds both for the sponge and this optimization

T/Key with the Sponge Construction: Security

Using generic composition: (assuming $q_{off} \geq q_{on}$)

$$\mathbf{Adv}^{\mathrm{T/Key}}_{\mathrm{Sponge}}(q_{off}, q_{on}, 1) = \mathcal{O}\Bigg(\underbrace{\frac{q_{off}}{2^{n+s}} + \frac{q_{on}}{2^n}}_{\mathbf{Adv}^{\mathrm{T/Key}}_{\mathcal{RO}}} + \underbrace{\frac{q_{off}^2}{2^c}}_{\mathbf{Adv}^{\mathrm{iff}}_{\mathrm{Sponge}}}\Bigg)$$

$\Longrightarrow$ Minimum permutation size: $b = 256$

T/Key with the Sponge Construction: Security

Using generic composition: (assuming $q_{off} \geq q_{on}$)

$$\mathbf{Adv}^{\mathrm{T/Key}}_{\mathrm{Sponge}}(q_{off}, q_{on}, 1) = \mathcal{O}\left(\underbrace{\frac{q_{off}}{2^{n+s}} + \frac{q_{on}}{2^{n}}}_{\mathbf{Adv}^{\mathrm{T/Key}}_{\mathcal{RO}}} + \underbrace{\frac{q_{off}^2}{2^{c}}}_{\mathbf{Adv}^{\mathrm{iff}}_{\mathrm{Sponge}}}\right)$$

$\Longrightarrow$ Minimum permutation size: $b = 256$

We derive:

$$\mathbf{Adv}^{\mathrm{T/Key}}_{\mathrm{Sponge}}(q_{off}, q_{on}, 1) = \mathcal{O}\left(\frac{Kq_{off}}{2^{n}} + \frac{Kq_{off}}{2^{c}} + \min\left(\frac{q_{on}}{2^{n-r}}; \frac{q_{off}\, q_{on}}{2^{c}}\right)\right)$$

$\Longrightarrow$ Password size increased to $n \geq 149$ ...

... but permutation size can be lowered to $b = 150$

$\Longrightarrow$ Could be instantiated with, e.g., Spongent permutation [BKL+11] ($b = 176, c = 150, r = 26$)

T/Key with the Sponge Construction: Proof Idea

$$\mathbf{Adv}^{\mathrm{T/Key}}_{\mathrm{Sponge}}(q_{off}, q_{on}, 1) = \mathcal{O}\left(\frac{Kq_{off}}{2^n} + \frac{Kq_{off}}{2^c} + \min\left(\frac{q_{on}}{2^{n-r}}; \frac{q_{off}\, q_{on}}{2^c}\right)\right)$$

Building blocks:

T/Key with the Sponge Construction: Proof Idea

$$\mathbf{Adv}^{\mathrm{T/Key}}_{\mathrm{Sponge}}(q_{off}, q_{on}, 1) = \mathcal{O}\left(\frac{Kq_{off}}{2^n} + \frac{Kq_{off}}{2^c} + \min\left(\frac{q_{on}}{2^{n-r}}; \frac{q_{off}\, q_{on}}{2^c}\right)\right)$$

Building blocks:

- Preimage resistance of the sponge [LM22]:

$$\mathbf{Adv}^{\mathrm{epre}}_{\mathrm{Sponge}}(q) = \mathcal{O}\left(\frac{q}{2^n} + \min\left(\frac{q}{2^{n-r}}; \frac{q(q+1)}{2^c}\right)\right)$$

T/Key with the Sponge Construction: Proof Idea

$$\mathbf{Adv}^{\mathrm{T/Key}}_{\mathrm{Sponge}}(q_{off}, q_{on}, 1) = \mathcal{O}\left(\frac{Kq_{off}}{2^n} + \frac{Kq_{off}}{2^c} + \min\left(\frac{q_{on}}{2^{n-r}}; \frac{q_{off}\, q_{on}}{2^c}\right)\right)$$

Building blocks:

- Preimage resistance of the sponge [LM22]:

$$\mathbf{Adv}^{\mathrm{epre}}_{\mathrm{Sponge}}(q) = \mathcal{O}\left(\frac{q}{2^n} + \min\left(\frac{q}{2^{n-r}}; \frac{q(q+1)}{2^c}\right)\right)$$

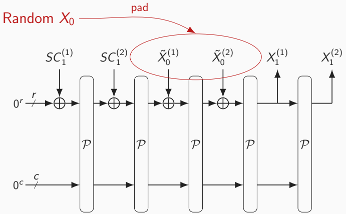
- PRF security of the outer keyed sponge with key size n [ADMV15, NY16, Men18]: (assuming $n \leq b$)

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{OKS}}(M, N) = \mathcal{O}\left(\frac{NM}{2^c} + \frac{N}{2^n}\right)$$

where M denotes the online complexity and N the offline complexity

T/Key with the Sponge Construction: Proof Idea (cont'd)

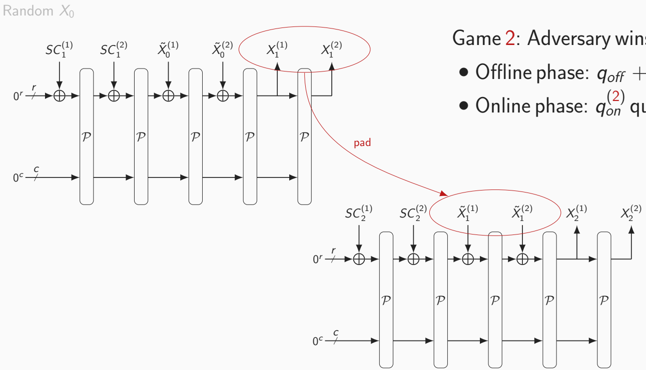
Simplified idea: the game can be decomposed into K different games



Game 1: Adversary wins if it finds a preimage of X_1

- Offline phase: $q_{off} + \sum_{k=2}^{K} q_{on}^{(k)}$ queries
- Online phase: $q_{on}^{(1)}$ queries

Simplified idea: the game can be decomposed into K different games

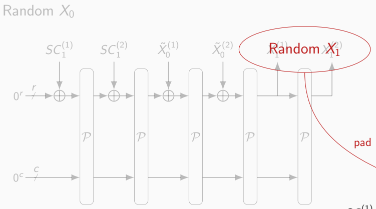


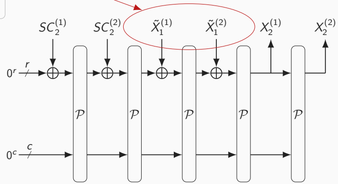
Game 2: Adversary wins if it finds a preimage of X_2

- Offline phase: $q_{off} + \sum_{k=3}^{K} q_{on}^{(k)}$ queries
- Online phase: $q_{on}^{(2)}$ queries

T/Key with the Sponge Construction: Proof Idea (cont'd)

Simplified idea: the game can be decomposed into K different games

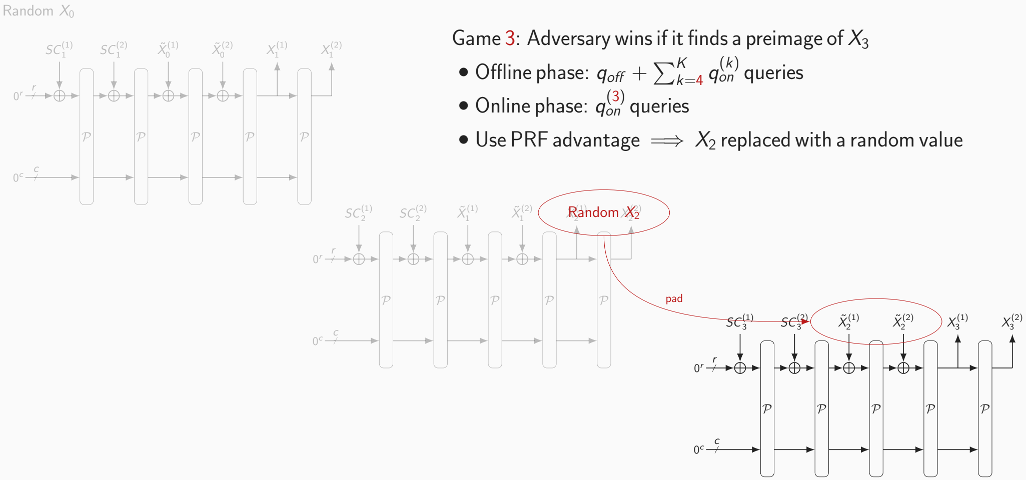




Game 2: Adversary wins if it finds a preimage of X_2

- Offline phase: $q_{off} + \sum_{k=3}^{K} q_{on}^{(k)}$ queries
- Online phase: $q_{on}^{(2)}$ queries
- Use PRF advantage $\Longrightarrow$ X_1 replaced with a random value

Simplified idea: the game can be decomposed into K different games

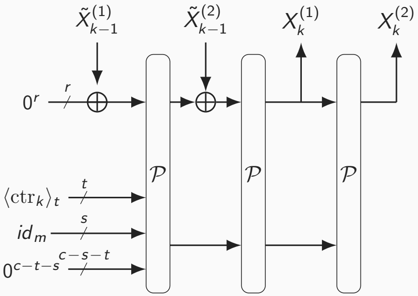


Game 3: Adversary wins if it finds a preimage of X_3

- Offline phase: $q_{off} + \sum_{k=4}^{K} q_{on}^{(k)}$ queries
- Online phase: $q_{on}^{(3)}$ queries
- Use PRF advantage $\implies$ X_2 replaced with a random value

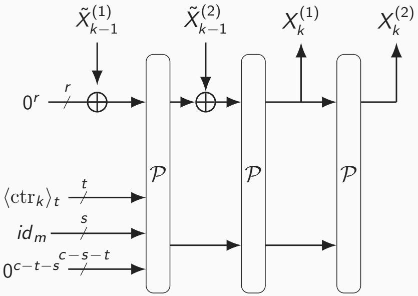
T/Key with a Truncated Permutation

T/Key with a Truncated Permutation



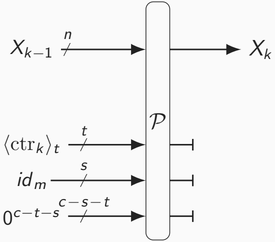
- Using a sponge?

T/Key with a Truncated Permutation



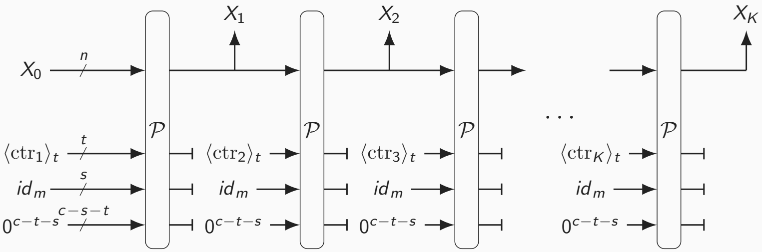
- Using a sponge?
- In most cases, the permutation is large enough to absorb everything at once

T/Key with a Truncated Permutation



- Using a sponge?
- In most cases, the permutation is large enough to absorb everything at once
- Construction behaves as truncated permutation

T/Key with a Truncated Permutation



- Using a sponge?
- In most cases, the permutation is large enough to absorb everything at once
- Construction behaves as truncated permutation
- Hash chain becomes truncated permutation chain

Security of T/Key with a Truncated Permutation

- Using generic composition with indifferentiability:

$$\mathbf{Adv}^{\mathrm{T/Key}}_{\mathrm{TruncP}}(q_{off}, q_{on}, 1) = \mathcal{O}\Bigg(\underbrace{\frac{q_{off}}{2^{n+s}} + \frac{q_{on}}{2^{n}}}_{\mathbf{Adv}^{\mathrm{T/Key}}_{\mathcal{RO}}} + \underbrace{\frac{q_{off}^{3/2}}{2^{\frac{2b-n}{2}}} + \frac{q_{off}}{2^{b-(n+t)}}}_{\mathbf{Adv}^{\mathrm{iff}}_{\mathrm{TruncP}}}\Bigg)$$

Security of T/Key with a Truncated Permutation

- Using generic composition with indifferentiability:

$$\mathbf{Adv}^{\mathrm{T/Key}}_{\mathrm{TruncP}}(q_{off}, q_{on}, 1) = \mathcal{O}\left(\underbrace{\frac{q_{off}}{2^{n+s}} + \frac{q_{on}}{2^n}}_{\mathbf{Adv}^{\mathrm{T/Key}}_{\mathcal{RO}}} + \underbrace{\frac{q_{off}^{3/2}}{2^{\frac{2b-n}{2}}} + \frac{q_{off}}{2^{b-(n+t)}}}_{\mathbf{Adv}^{\mathrm{iff}}_{\mathrm{TruncP}}} \right)$$

- We derive:

$$\mathbf{Adv}^{\mathrm{T/Key}}_{\mathrm{TruncP}}(q_{off}, q_{on}, M) = \mathcal{O}\left(\mathbf{Adv}^{\mathrm{T/Key}}_{\mathcal{RO}}(q_{off}, q_{on}, M) + \frac{KM \cdot q_{off}}{2^b} + \max\left(\frac{KM}{2^n}; 1\right) \cdot \frac{q_{on}}{2^c} \right)$$

- The bound is tight

Security of T/Key with a Truncated Permutation

- Using generic composition with indifferentiability:

$$\mathbf{Adv}^{\mathrm{T/Key}}_{\mathrm{TruncP}}(q_{off}, q_{on}, 1) = \mathcal{O}\Bigg(\underbrace{\frac{q_{off}}{2^{n+s}} + \frac{q_{on}}{2^n}}_{\mathbf{Adv}^{\mathrm{T/Key}}_{\mathcal{RO}}} + \underbrace{\frac{q_{off}^{3/2}}{2^{\frac{2b-n}{2}}} + \frac{q_{off}}{2^{b-(n+t)}}}_{\mathbf{Adv}^{\mathrm{iff}}_{\mathrm{TruncP}}} \Bigg)$$

- We derive:

$$\mathbf{Adv}^{\mathrm{T/Key}}_{\mathrm{TruncP}}(q_{off}, q_{on}, M) = \mathcal{O}\left(\mathbf{Adv}^{\mathrm{T/Key}}_{\mathcal{RO}}(q_{off}, q_{on}, M) + \frac{KM \cdot q_{off}}{2^b} + \max\left(\frac{KM}{2^n}; 1\right) \cdot \frac{q_{on}}{2^c} \right)$$

- The bound is tight
- Assume we want password sizes of $n = 100$, $q_{off} \ll 2^{128}$, $q_{on} \ll 2^{100}$:
  - Generic composition indicates we need $b \geq 260$
  - Our bound indicates we need $b \geq 200$ as long as $M \ll 2^{40}$

Conclusion

Conclusion

We analyzed the security of hash chain based password systems:

- **Refined model** that distinguishes offline vs. online complexity
- **Security proofs** with a random oracle, sponge, and truncated permutation:
  - Shows that truncated permutations work for most use cases
  - With truncated permutation, password size can be lowered to $n = 100$
  - Improved understanding of the preimage resistance of cascaded sponge evaluations
- Results hold in the random oracle/permutation model

Conclusion

We analyzed the security of hash chain based password systems:

- **Refined model** that distinguishes offline vs. online complexity
- **Security proofs** with a random oracle, sponge, and truncated permutation:
  - Shows that truncated permutations work for most use cases
  - With truncated permutation, password size can be lowered to $n = 100$
  - Improved understanding of the preimage resistance of cascaded sponge evaluations
- Results hold in the random oracle/permutation model

References i

Elena Andreeva, Joan Daemen, Bart Mennink, and Gilles Van Assche.
Security of Keyed Sponge Constructions Using a Modular Proof Approach.
In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 364–384. Springer, 2015.

Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche.
Sponge functions.
Ecrypt Hash Workshop 2007, May 2007.

Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche.
On the Indifferentiability of the Sponge Construction.
In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008.

Andrey Bogdanov, Miroslav Knezevic, Gregor Leander, Deniz Toz, Kerem Varici, and Ingrid Verbauwhede.
spongent: A Lightweight Hash Function.
In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan,*

September 28 - October 1, 2011. Proceedings, volume 6917 of *Lecture Notes in Computer Science*, pages 312–325. Springer, 2011.

Neil Haller.
The S/KEY One-Time Password System.
Request for Comments (RFC) 1760, February 1995.

Dmitry Kogan, Nathan Manohar, and Dan Boneh.
T/Key: Second-Factor Authentication From Secure Hash Chains.
In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 983–999. ACM, 2017.

Leslie Lamport.
Password Authentification with Insecure Communication.
Commun. ACM, 24(11):770–772, 1981.

Charlotte Lefevre and Bart Mennink.
Tight Preimage Resistance of the Sponge Construction.
In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 185–204. Springer, 2022.

Bart Mennink.
Key Prediction Security of Keyed Sponges.
IACR Trans. Symmetric Cryptol., 2018(4):128–149, 2018.

- D. M'Raihi, S. Machani, M. Pei, and J. Rydell.
  TOTP: Time-Based One-Time Password Algorithm.
  Request for Comments (RFC) 6238, May 2011.

- Yusuke Naito and Kan Yasuda.
  New Bounds for Keyed Sponges with Extendable Output: Independence Between Capacity and Message Length.
  In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2016.