CHIP and CRISP

Password-based key exchange: Storage hardening beyond the client-server setting

Cas Cremers, Moni Naor, Shahar Paz, **Eyal Ronen**




https://ia.cr/2020/529

First, let's get this out of the way...

No, the password is not dead

Websites, IoT, Wi-Fi, TLS 1.3, ….

Password use-cases

- **Authentication**
  - Login to website or server
- Creating a **secure channel**
  - Symmetric: One-to-One
  - Asymmetric: Client-to-Server
- **But what about Many-to-Many?**
  - Can’t we just use multiple one-to-one connections?

Typical Use Case
Smart Home Network

Smart home network

Computer - Fully patched Linux machine

Smart home network


Smart lock - Open front door on command from network

Smart home network

Thermostat - Bricked by vendor

Will be discarded with all credentials in persistent memory

Smart home network

Tablet - Android 8.0, last security patch January 2019

A proud member of 8 different major botnets

Smart home network

Router - Will be replaced next month (new provider)

Smart home network

Multiple visiting smartphones

The Wi-Fi Solution

All devices store a copy of the password

One compromise to rule them all !

Challenges in the Many-to-many setting

- **One password, many users/devices**
    - Source authentication
    - Revocation of specific users

Challenges in the Many-to-many setting

- **One password, many users/devices**
  - Source authentication
  - Revocation of specific users
- **Dynamic network topology**
  - Bootstrapping of new devices
  - Support for replacement of existing entities (e.g., replace hardware of Wi-Fi access point)

Challenges in the Many-to-many setting

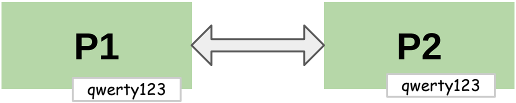
- **One password, many users/devices**
  - Source authentication
  - Revocation of specific users
- **Dynamic network topology**
  - Bootstrapping of new devices
  - Support for replacement of existing entities (e.g., replace hardware of Wi-Fi access point)
- **Asynchronous and offline password input**
  - No communication for setup and key generation phases
  - No shared randomness
  - No trusted third party or PKI

Related work

OPAQUE/saPAKE: hardening *servers*

[Jarecki S, Krawczyk H, Xu J ‘18]

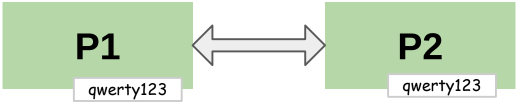
PAKE: Password Authenticated Key Exchange [BM’92]
Password never stored



OPAQUE/saPAKE: hardening *servers*

[Jarecki S, Krawczyk H, Xu J ‘18]

PAKE: Password Authenticated Key Exchange [BM’92]
Password never stored
Or stored in plaintext on both sides

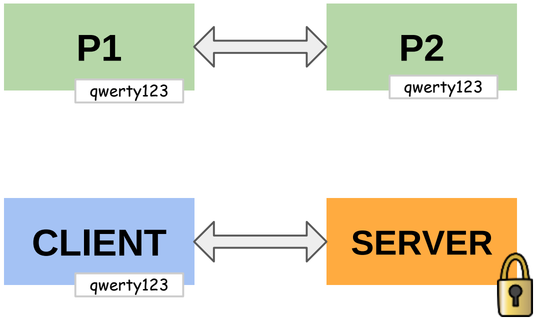


OPAQUE/saPAKE: hardening *servers*

[Jarecki S, Krawczyk H, Xu J '18]

PAKE: Password Authenticated Key Exchange [BM'92]
Password never stored
Or stored in plaintext on both sides

aPAKE: Asymmetric PAKE [BM'93]
Password not in plaintext on server



OPAQUE/saPAKE: hardening *servers*

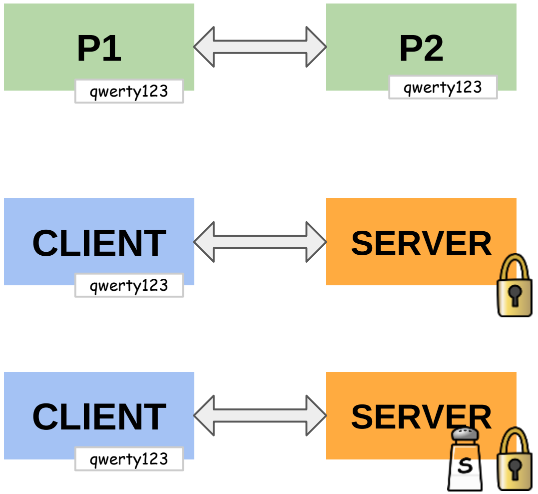
[Jarecki S, Krawczyk H, Xu J ‘18]

PAKE: Password Authenticated Key Exchange [BM’92]
Password never stored
Or stored in plaintext on both sides

aPAKE: Asymmetric PAKE [BM’93]
Password not in plaintext on server

saPAKE: Strong asymmetric PAKE [JKX’18]
Password storage on server prevents pre-computation

E.g. OPAQUE



OPAQUE/saPAKE: hardening *servers*

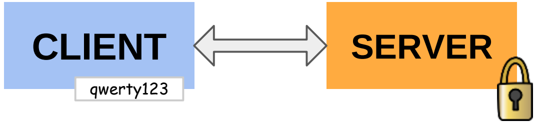
[Jarecki S, Krawczyk H, Xu J ‘18

Problem: (s)aPAKE techniques...

- **Require the password in plaintext on one side (the client)**
- **Do not work in the symmetric setting (eg Wifi)**

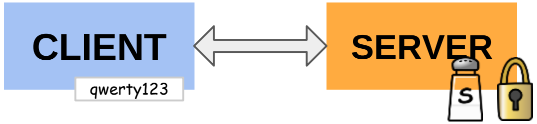
PAKE: Password Authenticated Key Exc
Password never stored
Or stored in plaintext on both sides

aPAKE: Asymmetric PAKE [BM’93]
Password not in plaintext on server



saPAKE: Strong asymmetric PAKE [JKX’18]
Password storage on server prevents pre-computation

E.g. OPAQUE



CHIP & CRISP!

We propose techniques to **protect all parties**

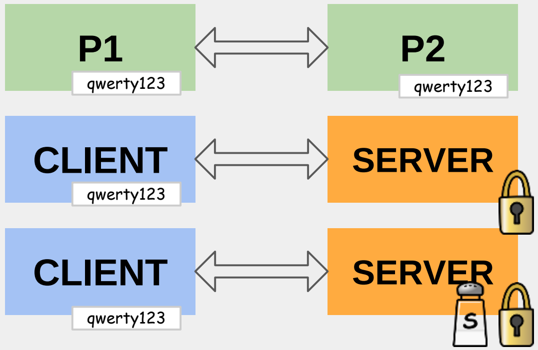
OLD

PAKE: Password Authenticated Key Exchange
Password in plaintext on both sides

aPAKE: Asymmetric PAKE
Password not in plaintext on server

saPAKE: Strong asymmetric PAKE
Password storage on server prevents pre-computation

OPAQUE



We propose techniques to **protect all parties**

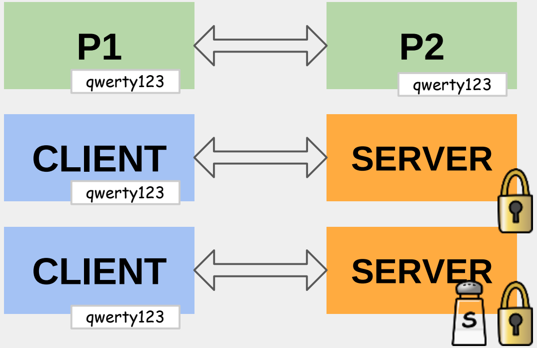
OLD

PAKE: Password Authenticated Key Exchange
Password in plaintext on both sides

aPAKE: Asymmetric PAKE
Password not in plaintext on server

saPAKE: Strong asymmetric PAKE
Password storage on server prevents pre-computation

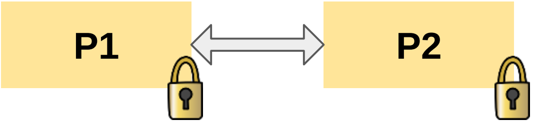
OPAQUE



NEW!

CHIP

iPAKE
Password not in plaintext at any party
Compromising P1 *only* allows impersonating P1, not P2!



We propose techniques to **protect all parties**

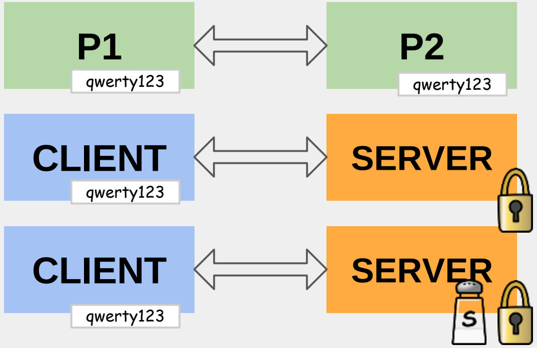
OLD

PAKE: Password Authenticated Key Exchange
Password in plaintext on both sides

aPAKE: Asymmetric PAKE
Password not in plaintext on server

saPAKE: Strong asymmetric PAKE
Password storage on server prevents pre-computation

OPAQUE



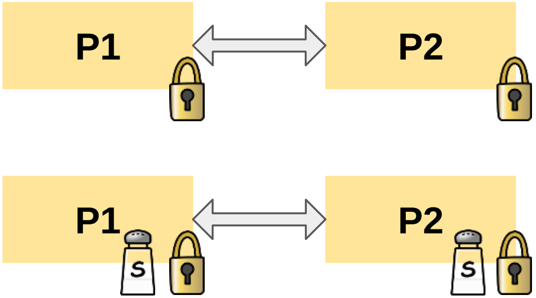
NEW!

iPAKE
Password not in plaintext at any party
Compromising P1 *only* allows impersonating P1, not P2!

CHIP

siPAKE
All password storage prevents pre-computation

CRISP



Realizing iPAKE and siPAKE

- We achieve iPAKE and siPAKE by using techniques from identity-based key exchange/agreement

Realizing iPAKE and siPAKE

- We achieve iPAKE and siPAKE by using techniques from identity-based key exchange/agreement
- We only use the underlying ideas

Realizing iPAKE and siPAKE

- We achieve iPAKE and siPAKE by using techniques from identity-based key exchange/agreement
- We only use the underlying ideas
  - We **do not need a trusted key generation center** or any other third party

Realizing iPAKE and siPAKE

- We achieve iPAKE and siPAKE by using techniques from identity-based key exchange/agreement
- We only use the underlying ideas
  - We **do not need a trusted key generation center** or any other third party
  - We **do not need unique identities**, but instead use abstract tags to bind the password storage to

Realizing iPAKE and siPAKE

- We achieve iPAKE and siPAKE by using techniques from identity-based key exchange/agreement
- We only use the underlying ideas
    - We **do not need a trusted key generation center** or any other third party
    - We **do not need unique identities**, but instead use abstract tags to bind the password storage to
        - Can choose to bind storage to **identities**, but also to **roles**, **unique devices identifiers**, etc.
        - Can have multiple devices **sharing the same tag**

Example: CRISP

Pre-Computation Resistance

- $H(pw)$
  - Vulnerable to pre-computation
  - Reverse lookup $H^{-1}[\cdot]$

Pre-Computation Resistance

- $H(pw)$
- $x, H(pw, x)$
  - Salted hash
  - Pre-computation resistant
  - Shared key without shared randomness?
    - $x, H(pw, x)$ vs. $y, H(pw, y)$
    - $H(\cdot)$ is Random Oracle
    - Without pw, cannot compute $H(pw, y)$ from $H(pw, x)$
    - Needs one way function with some kind of structure

Pre-Computation Resistance

- $H(pw)$
- $x, H(pw, x)$
- $x,\ g^{H(pw)\cdot x}$
  - Vulnerable to pre-computation
  - Pre-compute T: $g^{H(pw')} \mapsto pw'$
  - $\mathrm{T}\left[\left(g^{H(pw)\cdot x}\right)^{1/x}\right] = \mathrm{T}\left[g^{H(pw)}\right] = pw$
  - pw and x can be separated

Pre-Computation Resistance

- $H(pw)$
- $x, H(pw, x)$
- $x,\ g^{H(pw)\cdot x}$
- $g^x, g^{H(pw)\cdot x}$
    - Pre-computation resistant
    - Oracle Hashing [Can'97]
    - Salted Tight OWF [BJX'19]
    - Requires Pairing…

Pre-Computation Resistance

- $H(pw)$
- $x, H(pw, x)$
- $x,\ g^{H(pw)\cdot x}$
- $g^x, g^{H(pw)\cdot x}$
- $g^x, \widehat{H}(pw)^x$
  - Pre-computation resistant
  - Pairing + Hash-to-Group
  - Offline brute force cost is pairing - $\hat{e}\left(\widehat{H}(pw'), g^x\right) \stackrel{?}{=} \hat{e}\left(\widehat{H}(pw)^x, g\right)$

Password File Generation

P_i

$x_i \xleftarrow{R} Z_q^*$

$A_i \leftarrow g_1^{x_i}$

$B_i \leftarrow \widehat{H}_1(pw)^{x_i}$

$\boldsymbol{C_i} \leftarrow \boldsymbol{\widehat{H}_2}("Alice")^{\boldsymbol{x_i}}$

$\langle \text{"Alice"}, \boldsymbol{A_i}, \boldsymbol{B_i}, \boldsymbol{C_i} \rangle$

P_j

$x_j \xleftarrow{R} Z_q^*$

$A_j \leftarrow g_1^{x_j}$

$B_j \leftarrow \widehat{H}_1(pw)^{x_j}$

$\boldsymbol{C_j} \leftarrow \boldsymbol{\widehat{H}_2}("Bob")^{\boldsymbol{x_j}}$

$\langle \text{"Bob"}, \boldsymbol{A_j}, \boldsymbol{B_j}, \boldsymbol{C_j} \rangle$

Password File Generation

P_i

$$x_i \xleftarrow{R} Z_q^*$$
$$A_i \leftarrow g_1^{x_i}$$
$$B_i \leftarrow \widehat{H}_1(pw)^{x_i}$$
$$\boldsymbol{C_i} \leftarrow \widehat{\boldsymbol{H}}_2(\text{"}Alice\text{"})^{\boldsymbol{x_i}}$$
$$\langle \text{"Alice"}, \boldsymbol{A_i}, \boldsymbol{B_i}, \boldsymbol{C_i} \rangle$$

P_j

$$x_j \xleftarrow{R} Z_q^*$$
$$A_j \leftarrow g_1^{x_j}$$
$$B_j \leftarrow \widehat{H}_1(pw)^{x_j}$$
$$\boldsymbol{C_j} \leftarrow \widehat{\boldsymbol{H}}_2(\text{"}Bob\text{"})^{\boldsymbol{x_j}}$$
$$\langle \text{"Bob"}, \boldsymbol{A_j}, \boldsymbol{B_j}, \boldsymbol{C_j} \rangle$$

Password File Generation

P_i

$x_i \xleftarrow{R} Z_q^*$

$A_i \leftarrow g_1^{x_i}$

$B_i \leftarrow \hat{H}_1(pw)^{x_i}$

$\boldsymbol{C_i \leftarrow \hat{H}_2(\text{"Alice"})^{x_i}}$

$\langle \text{"Alice"}, \boldsymbol{A_i}, \boldsymbol{B_i}, \boldsymbol{C_i} \rangle$

P_j

$x_j \xleftarrow{R} Z_q^*$

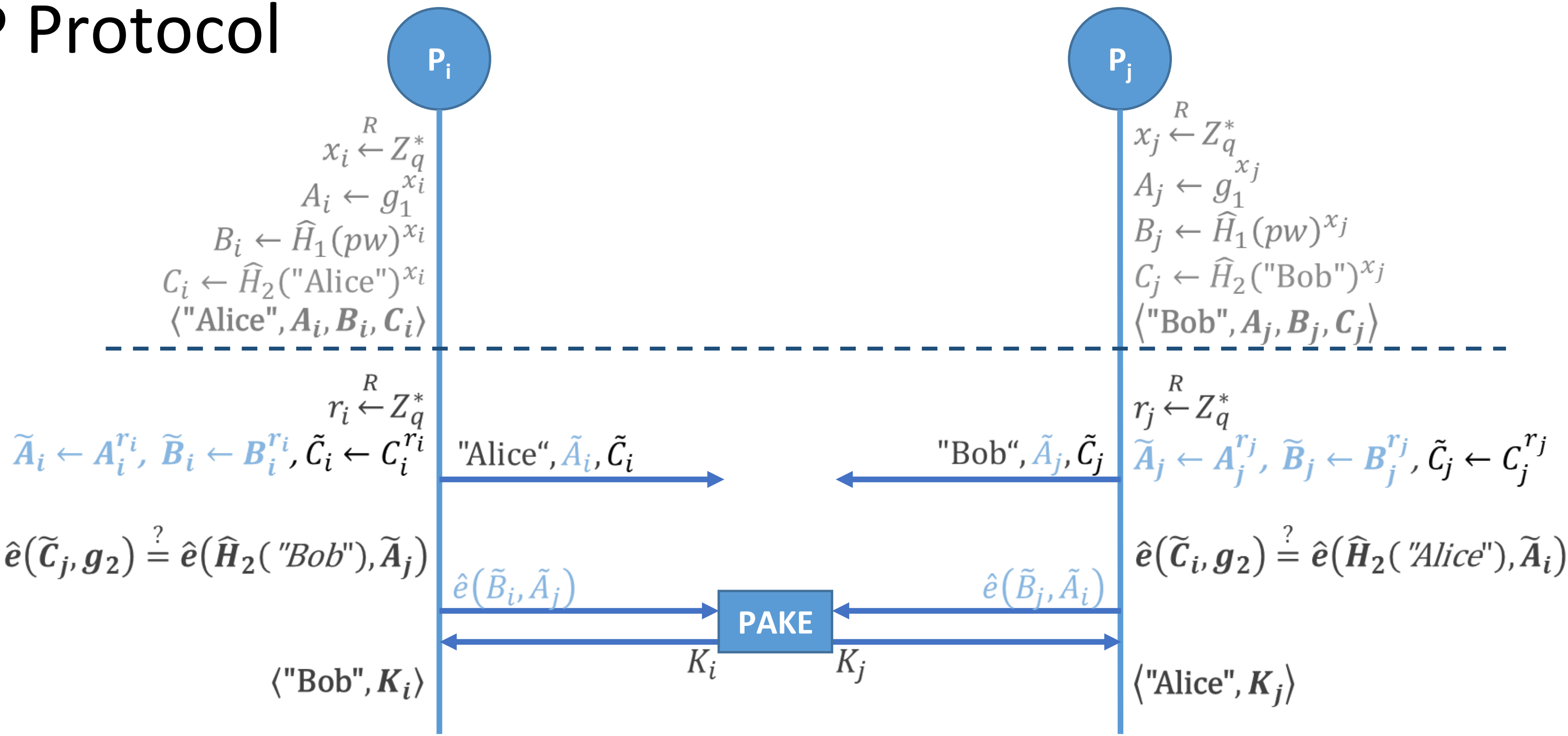
$A_j \leftarrow g_1^{x_j}$

$B_j \leftarrow \hat{H}_1(pw)^{x_j}$

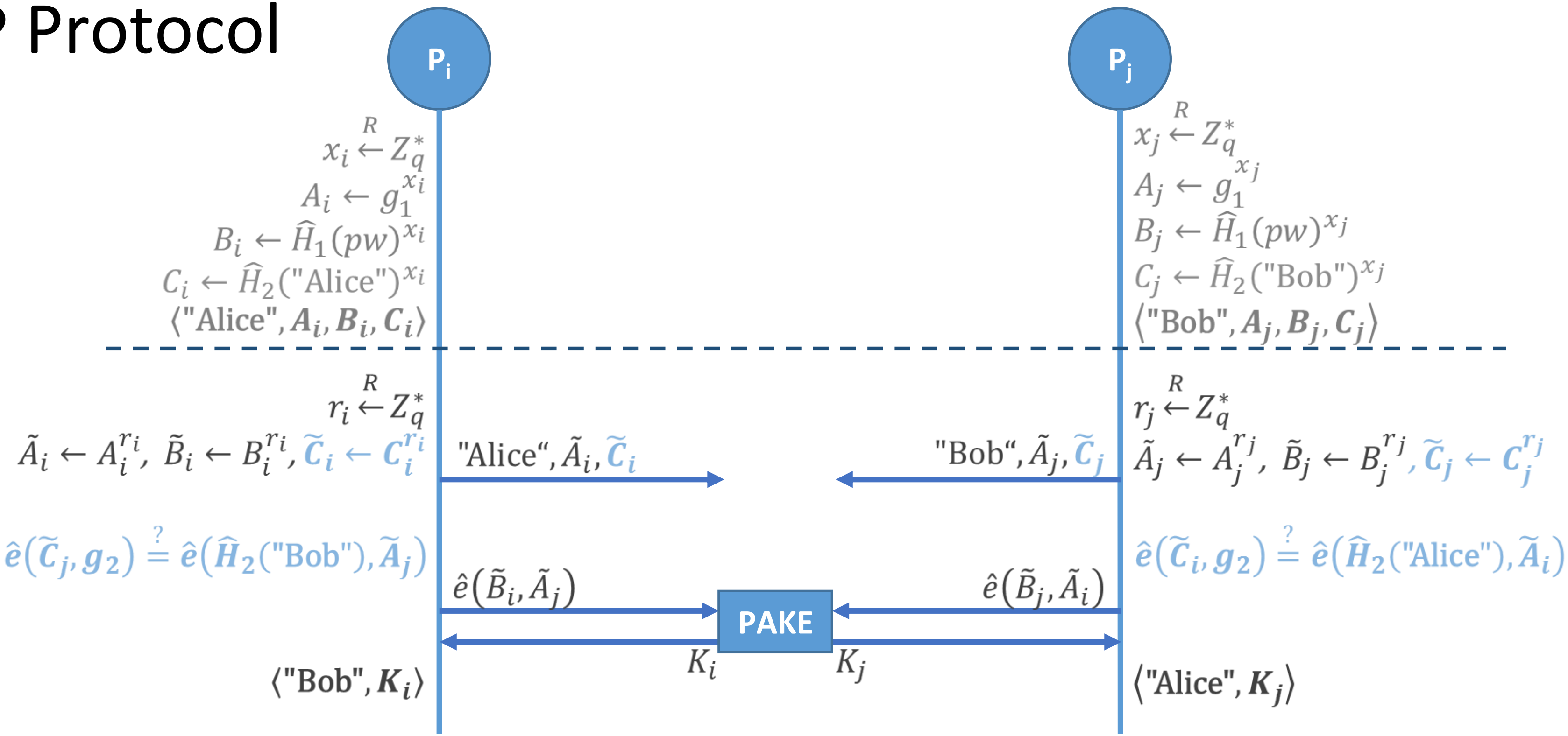
$\boldsymbol{C_j \leftarrow \hat{H}_2(\text{"Bob"})^{x_j}}$

$\langle \text{"Bob"}, \boldsymbol{A_j}, \boldsymbol{B_j}, \boldsymbol{C_j} \rangle$

CRISP Protocol



CRISP Protocol



Performance Comparison

	CPace	SAE	CHIP	OPAQUE	CRISP
CPU time (ms)	0.2	>1.3	0.6	0.6	4.1
Communication rounds	1	2	2	2	2
Security notion	PAKE	*none*	iPAKE	saPAKE	siPAKE

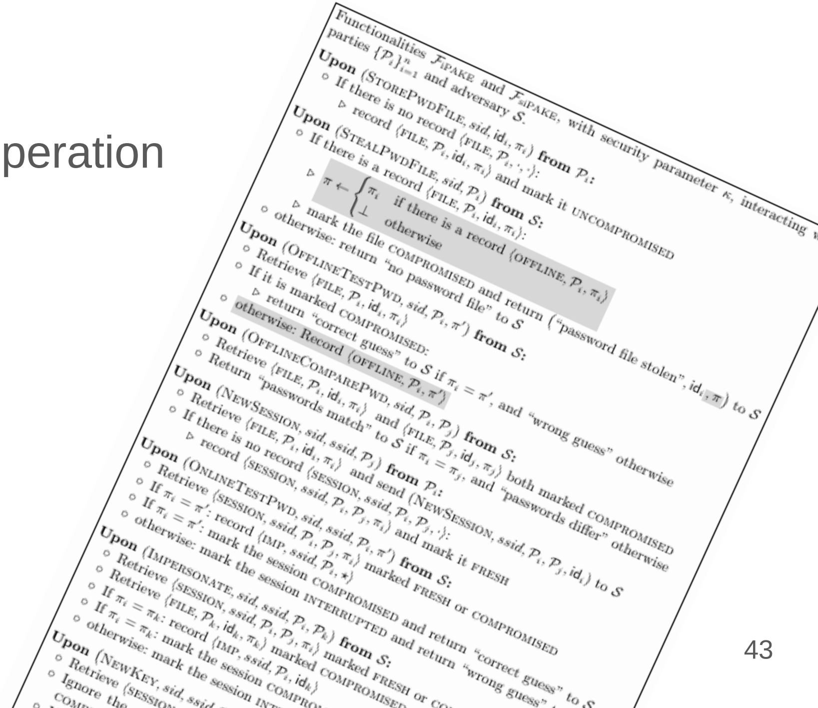
Low overhead, suitable for Wi-Fi and IoT networks

Several suggestion for optimizing CRISP

Code available at: https://github.com/shapaz/CRISP

Security

- We provide a UC ideal definition for iPAKE and siPAKE
- We prove CHIP under ROM
- We prove CRISP under GGM+ROM
  - Prove cost password guess is a pairing operation


Functionalities $\mathcal{F}_{\text{iPAKE}}$ and $\mathcal{F}_{\text{siPAKE}}$, with security parameter κ, interacting w
parties $\{\mathcal{P}_i\}_{i=1}^n$ and adversary $\mathcal{S}$.

Upon (StorePwdFile, sid, id_i, π_i) **from** $\mathcal{P}_i$:
∘ If there is no record ⟨FILE, $\mathcal{P}_i$, ·, ·⟩:
▷ record ⟨FILE, $\mathcal{P}_i$, id_i, π_i⟩ and mark it UNCOMPROMISED

Upon (StealPwdFile, sid, $\mathcal{P}_i$) **from** $\mathcal{S}$:
∘ If there is a record ⟨FILE, $\mathcal{P}_i$, id_i, π_i⟩:
▷ $\pi \leftarrow \begin{cases}\pi_i & \text{if there is a record } \langle \text{OFFLINE}, \mathcal{P}_i, \pi_i\rangle \\ \perp & \text{otherwise}\end{cases}$
▷ mark the file COMPROMISED and return ("password file stolen", id_i, π) to $\mathcal{S}$
∘ otherwise: return "no password file" to $\mathcal{S}$

Upon (OfflineTestPwd, sid, $\mathcal{P}_i$, π') **from** $\mathcal{S}$:
∘ Retrieve ⟨FILE, $\mathcal{P}_i$, id_i, π_i⟩
∘ If it is marked COMPROMISED:
▷ return "correct guess" to $\mathcal{S}$ if $\pi_i = \pi'$, and "wrong guess" otherwise
∘ otherwise: Record ⟨OFFLINE, $\mathcal{P}_i$, π'⟩

Upon (OfflineComparePwd, sid, $\mathcal{P}_i$, $\mathcal{P}_j$) **from** $\mathcal{S}$:
∘ Retrieve ⟨FILE, $\mathcal{P}_i$, id_i, π_i⟩ and ⟨FILE, $\mathcal{P}_j$, id_j, π_j⟩ both marked COMPROMISED
∘ Return "passwords match" to $\mathcal{S}$ if $\pi_i = \pi_j$, and "passwords differ" otherwise

Upon (NewSession, sid, $ssid$, $\mathcal{P}_j$) **from** $\mathcal{P}_i$:
∘ Retrieve ⟨FILE, $\mathcal{P}_i$, id_i, π_i⟩ and send (NewSession, $ssid$, $\mathcal{P}_i$, $\mathcal{P}_j$, id_i) to $\mathcal{S}$
∘ If there is no record ⟨SESSION, $ssid$, $\mathcal{P}_i$, $\mathcal{P}_j$, ·⟩:
▷ record ⟨SESSION, $ssid$, $\mathcal{P}_i$, $\mathcal{P}_j$, π_i⟩ and mark it FRESH

Upon (OnlineTestPwd, sid, $ssid$, $\mathcal{P}_i$, π') **from** $\mathcal{S}$:
∘ Retrieve ⟨SESSION, $ssid$, $\mathcal{P}_i$, $\mathcal{P}_j$, π_i⟩ marked FRESH or COMPROMISED
∘ If $\pi_i = \pi'$: record ⟨IMP, $ssid$, $\mathcal{P}_i$, $\star$⟩
∘ If $\pi_i = \pi'$: mark the session COMPROMISED and return "correct guess" to $\mathcal{S}$
∘ otherwise: mark the session INTERRUPTED and return "wrong guess"

Upon (Impersonate, sid, $ssid$, $\mathcal{P}_i$, $\mathcal{P}_k$) **from** $\mathcal{S}$:
∘ Retrieve ⟨SESSION, $ssid$, $\mathcal{P}_i$, $\mathcal{P}_j$, π_i⟩ marked FRESH or CO
∘ Retrieve ⟨FILE, $\mathcal{P}_k$, id_k, π_k⟩ marked COMPROMISED
∘ If $\pi_i = \pi_k$: record ⟨IMP, $ssid$, $\mathcal{P}_i$, id_k⟩
∘ If $\pi_i = \pi_k$: mark the session COMPRO
∘ otherwise: mark the session INT

Upon (NewKey, sid, $ssid$
∘ Retrieve ⟨SESSION
∘ Ignore the

Open Questions

- Does siPAKE requires GGM?
- Can we have fine grained post-compromise password hardening?
- Optimal bound on the cost of brute-force attack?
- Two messages (s)iPAKE?

Conclusions

https://ia.cr/2020/529

https://github.com/shapaz/CRISP

CHIP and **CRISP**:

1. provide stronger guarantees for password storage to all parties, and
2. work in the symmetric setting!

CHIP and CRISP: Protecting All Parties Against Compromise through Identity-Binding PAKEs

Cas Cremers and Moni Naor and Shahar Paz and Eyal Ronen