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

![Diagram showing the PAKE protocol involving users P1 and P2 with the password qwerty123 exchanged between them.](image)
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

PAKE: Password Authenticated Key Exchange
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

Problem: (s)aPAKE techniques...
- Require the password in plaintext on one side (the client)
- Do not work in the symmetric setting (eg Wifi)

E.g. OPAQUE
CHIP & CRISP!
We propose techniques to **protect all parties**

**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
We propose techniques to **protect all parties**

**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

**NEW**  
**iPAKE**  
Password not in plaintext at any party  
Compromising P1 *only* allows impersonating P1, not P2!
We propose techniques to **protect all parties**

**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

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

**siPAKE**
All password storage prevents pre-computation
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 \( T: g^{H(pw)} \mapsto pw' \)
  - \( T \left[ (g^{H(pw)} \cdot x)^{1/x} \right] = T[g^{H(pw)}] = pw \)
  - \( pw \) 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

- $H(pw)$
- $x, H(pw, x)$
- $x, g^{H(pw)} \cdot x$
- $g^x, g^{H(pw)} \cdot x$
- $g^x, \tilde{H}(pw)^x$
  - Pre-computation resistant
  - Pairing + Hash-to-Group
  - Offline brute force cost is pairing - $\hat{e}(\tilde{H}(pw'), g^x) \neq \hat{e}(\tilde{H}(pw)^x, g)$
Password File Generation

\[
\begin{align*}
    x_i & \leftarrow \mathbb{Z}_q^* \\
    A_i & \leftarrow g_1^{x_i} \\
    B_i & \leftarrow \tilde{H}_1(pw)^{x_i} \\
    C_i & \leftarrow \tilde{H}_2("Alice")^{x_i} \\
    \langle "Alice", A_i, B_i, C_i \rangle
\end{align*}
\]

\[
\begin{align*}
    x_j & \leftarrow \mathbb{Z}_q^* \\
    A_j & \leftarrow g_1^{x_j} \\
    B_j & \leftarrow \tilde{H}_1(pw)^{x_j} \\
    C_j & \leftarrow \tilde{H}_2("Bob")^{x_j} \\
    \langle "Bob", A_j, B_j, C_j \rangle
\end{align*}
\]
Password File Generation

\[ x_i \leftarrow Z_q^* \]
\[ A_i \leftarrow g_1^{x_i} \]
\[ B_i \leftarrow \tilde{H}_1(pw)^{x_i} \]
\[ C_i \leftarrow \tilde{H}_2("Alice")^{x_i} \]
\[ \langle "Alice", A_i, B_i, C_i \rangle \]

\[ x_j \leftarrow Z_q^{x_j} \]
\[ A_j \leftarrow g_1^{x_j} \]
\[ B_j \leftarrow \tilde{H}_1(pw)^{x_j} \]
\[ C_j \leftarrow \tilde{H}_2("Bob")^{x_j} \]
\[ \langle "Bob", A_j, B_j, C_j \rangle \]
Password File Generation

\[ x_i \leftarrow Z_q^* \]
\[ A_i \leftarrow g_1^{x_i} \]
\[ B_i \leftarrow H_1(pw)^{x_i} \]
\[ C_i \leftarrow H_2("Alice")^{x_i} \]
\[ \langle "Alice", A_i, B_i, C_i \rangle \]

\[ x_j \leftarrow Z_q^* \]
\[ A_j \leftarrow g_1^{x_j} \]
\[ B_j \leftarrow H_1(pw)^{x_j} \]
\[ C_j \leftarrow H_2("Bob")^{x_j} \]
\[ \langle "Bob", A_j, B_j, C_j \rangle \]
CRISP Protocol

P_i

\[
\begin{align*}
x_i & \leftarrow Z_q^* \\
A_i & \leftarrow g_1^{x_i} \\
B_i & \leftarrow \tilde{H}_1(pw)^{x_i} \\
C_i & \leftarrow \tilde{H}_2("Alice")^{x_i} \\
\langle "Alice", A_i, B_i, C_i \rangle \\
\end{align*}
\]

\[
\begin{align*}
\tilde{A}_i & \leftarrow A_i^{r_i} \\
\tilde{B}_i & \leftarrow B_i^{r_i} \\
\tilde{C}_i & \leftarrow C_i^{r_i} \\
\end{align*}
\]

\[
\tilde{A}_i, \tilde{A}_i, \tilde{C}_i
\]

\[
\hat{e}(\tilde{C}_j, g_2) \overset{?}{=} \hat{e}(\tilde{H}_2("Bob"), \tilde{A}_j)
\]

\[
\hat{e}(\tilde{B}_i, \tilde{A}_i)
\]

PAKE

\[
\langle "Bob", K_i \rangle
\]

P_j

\[
\begin{align*}
x_j & \leftarrow Z_q^* \\
A_j & \leftarrow g_1^{x_j} \\
B_j & \leftarrow \tilde{H}_1(pw)^{x_j} \\
C_j & \leftarrow \tilde{H}_2("Bob")^{x_j} \\
\langle "Bob", A_j, B_j, C_j \rangle \\
\end{align*}
\]

\[
\begin{align*}
\tilde{A}_j & \leftarrow A_j^{r_j} \\
\tilde{B}_j & \leftarrow B_j^{r_j} \\
\tilde{C}_j & \leftarrow C_j^{r_j} \\
\end{align*}
\]

\[
\tilde{A}_j, \tilde{A}_j, \tilde{C}_j
\]

\[
\hat{e}(\tilde{C}_i, g_2) \overset{?}{=} \hat{e}(\tilde{H}_2("Alice"), \tilde{A}_i)
\]

\[
\hat{e}(\tilde{B}_j, \tilde{A}_j)
\]

\[
\langle "Alice", K_j \rangle
\]
CRISP Protocol

\[ x_i \leftarrow Z_q^* \]
\[ A_i \leftarrow g_1^{x_i} \]
\[ B_i \leftarrow H_1(pw)^{x_i} \]
\[ C_i \leftarrow H_2("Alice")^{x_i} \]
\[ \langle "Alice", A_i, B_i, C_i \rangle \]

\[ \tilde{A}_i \leftarrow A_i^{r_i} \]
\[ \tilde{B}_i \leftarrow B_i^{r_i} \]
\[ \tilde{C}_i \leftarrow C_i^{r_i} \]
\[ "Alice", \tilde{A}_i, \tilde{C}_i \]

\[ e(\tilde{C}_j, g_2) = e(H_2("Bob"), \tilde{A}_j) \]

\[ e(\tilde{B}_i, \tilde{A}_j) \]
\[ \langle "Bob", K_i \rangle \]

\[ e(\tilde{B}_j, \tilde{A}_i) \]
\[ K_i \]
\[ K_j \]

\[ e(\tilde{C}_i, g_2) = e(H_2("Alice"), \tilde{A}_i) \]
\[ \langle "Alice", K_j \rangle \]
Performance Comparison

<table>
<thead>
<tr>
<th></th>
<th>CPace</th>
<th>SAE</th>
<th>CHIP</th>
<th>OPAQUE</th>
<th>CRISP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU time (ms)</td>
<td>0.2</td>
<td>&gt;1.3</td>
<td>0.6</td>
<td>0.6</td>
<td>4.1</td>
</tr>
<tr>
<td>Communication rounds</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Security notion</td>
<td>PAKE</td>
<td>none</td>
<td>iPAKE</td>
<td>saPAKE</td>
<td>siPAKE</td>
</tr>
</tbody>
</table>

Low overhead, suitable for Wi-Fi and IoT networks

Several suggestions 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
Open Questions

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

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

https://ia.cr/2020/529
https://github.com/shapaz/CRISP