Multiple Passwords in WPA3: Use Cases & Initial Proposals

Mathy Vanhoef

Workshop on Password Authenticated Key Exchange and Password Security & Usability (PAKE'25). 7 February 2025, Luxembourg.

Funded by NGI Sargasso under the DecoyAuth project.

** These slides are slightly updated based on feedback after the presentation.*

Wi-Fi history

- 1999: WEP: completely broken
- 2003-2004: WPA1/2
  - Password-protected ‘home’ networks & Enterprise EAP authentication
  - Vulnerable to offline dictionary attacks (no forward secrecy)

Wi-Fi history

- 1999: WEP: completely broken
- 2003-2004: WPA1/2
  - **Password-protected 'home' networks** & Enterprise EAP authentication
  - Vulnerable to offline dictionary attacks (no forward secrecy)

Multiple WPA2 passwords

A single network name but multiple passwords

- Better user experience + less airtime overhead
- Use case: **guests get a different password**
  - Devices connect to same network, but are put in different VLANs
- Use case: **all users or devices get a different password**
  - Infer identity from used password, can again have different VLANs
  - Revoke/change individual passwords, e.g., hotels, employees,…
  - **Malicious insider** can’t create rogue clone of the network

Multi-password WPA2 in practice

Implemented by practically **all vendors**!

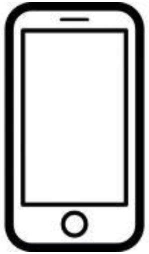
- Downside: network-side must loop through all passwords

- Nice alternative to have per-user credentials…
- …but without the hassle of certificates/usernames

Wi-Fi history

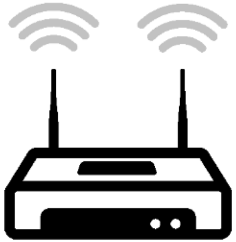
- 1991: WEP: completely broken
- 2003-2004: WPA1/2
  - Password-protected 'home' networks & Enterprise EAP authentication
  - Vulnerable to offline dictionary attacks (no forward secrecy)
- **2018: WPA3**
  - Uses the "Dragonfly" PAKE and is similar to SPEKE
  - Was vulnerable to "Dragonblood" side-channel attacks (now fixed)
  - Used in mesh networks too (hence symmetric PAKE)
  - We focus on elliptic curve variant

Dragonfly



Pick random r_A and m_A
$s_A = (r_A + m_A) \bmod q$
$E_A = -m_A\, P$

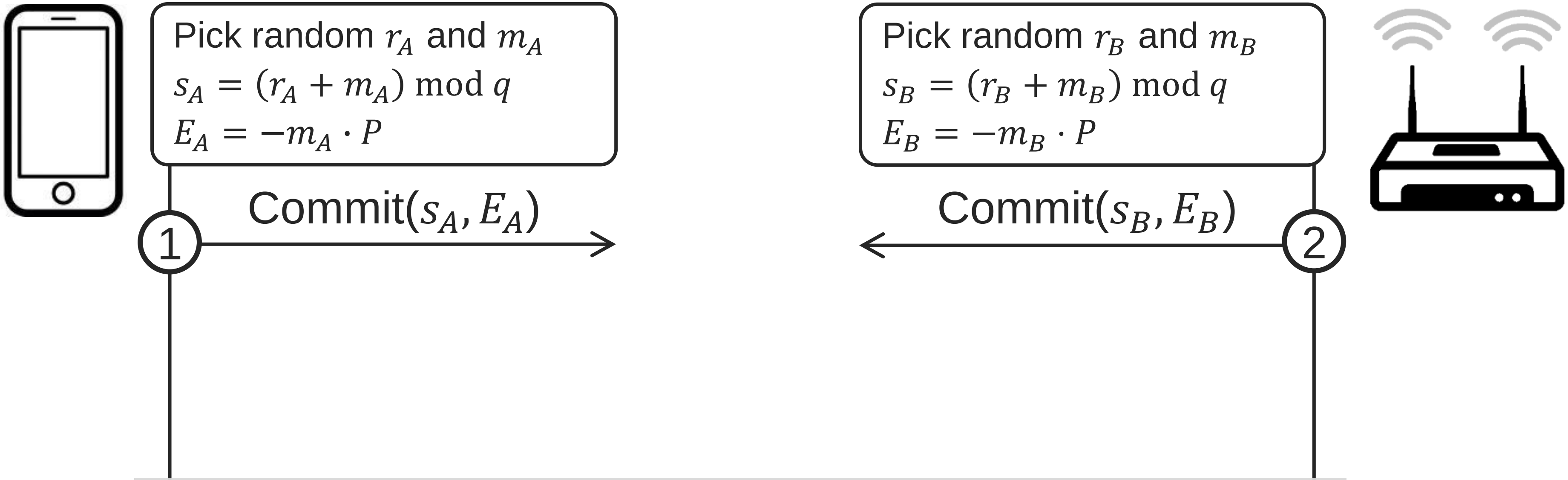
Pick random r_B and m_B
$s_B = (r_B + m_B) \bmod q$
$E_B = -m_B\, P$



Password is hashed to group element P

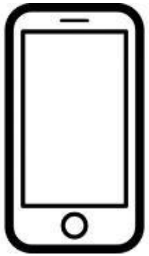
(Simplified Shallue Woestijne-Ulas)

Dragonfly



Could also have design without scalar s, it was added to avoid patent issues…

Dragonfly

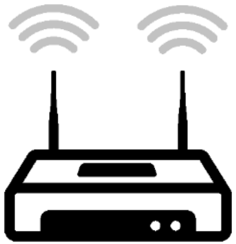


Pick random r_A and m_A

$s_A = (r_A + m_A) \bmod q$

$E_A = -m_A \cdot P$

① Commit(s_A, E_A) →



Pick random r_B and m_B

$s_B = (r_B + m_B) \bmod q$

$E_B = -m_B \cdot P$

② ← Commit(s_B, E_B)

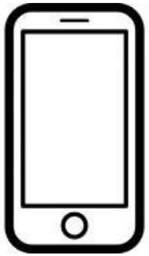
$$
\begin{aligned}
K &= r_A \cdot (s_B \cdot P + E_B) \\
&= r_A \cdot (r_B \cdot P + m_B \cdot P - m_B \cdot P) \\
&= r_A \cdot r_B \cdot P
\end{aligned}
$$

$\kappa = \mathrm{Hash}(K)$

$tr = (s_A, E_A, s_B, E_B)$

$c_A = \mathrm{HMAC}(\kappa, tr)$

Dragonfly



Pick random r_A and m_A

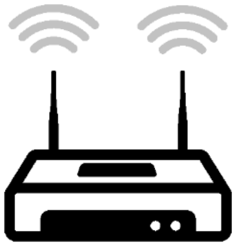
$s_A = (r_A + m_A) \bmod q$

$E_A = -m_A \cdot P$

Pick random r_B and m_B

$s_B = (r_B + m_B) \bmod q$

$E_B = -m_B \cdot P$



1 Commit(s_A, E_A) →

← Commit(s_B, E_B) 2

$K = r_A \cdot (s_B \cdot P + E_B) = \boldsymbol{r_A \cdot r_B \cdot P}$

$\kappa = \text{Hash}(K)$

$tr = (s_A, E_A, s_B, E_B)$

$c_A = \text{HMAC}(\kappa, tr)$

Dragonfly

Pick random r_A and m_A
$s_A = (r_A + m_A) \bmod q$
$E_A = -m_A \cdot P$

① Commit(s_A, E_A) →

Pick random r_B and m_B
$s_B = (r_B + m_B) \bmod q$
$E_B = -m_B \cdot P$

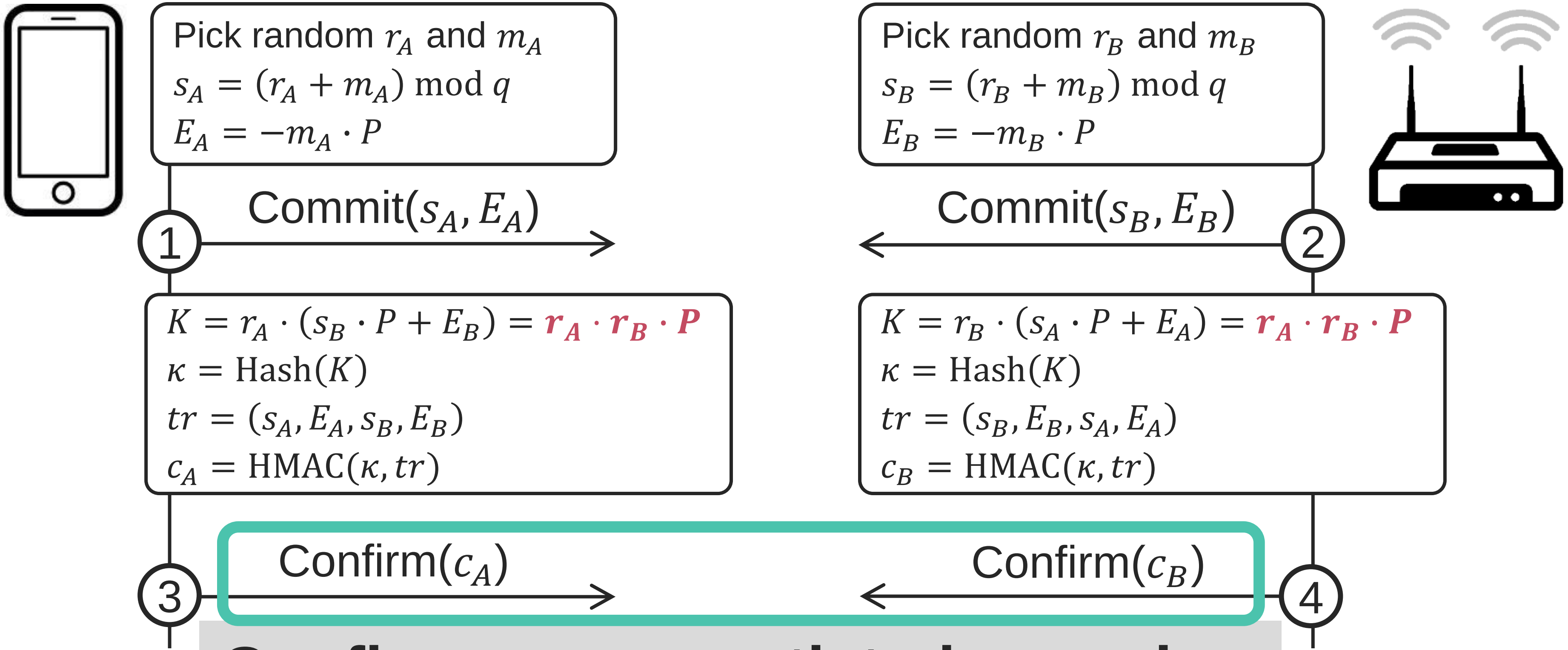
② ← Commit(s_B, E_B)

$K = r_A \cdot (s_B \cdot P + E_B) = \boldsymbol{r_A \cdot r_B \cdot P}$
$\kappa = \text{Hash}(K)$
$tr = (s_A, E_A, s_B, E_B)$
$c_A = \text{HMAC}(\kappa, tr)$

$K = r_B \cdot (s_A \cdot P + E_A) = \boldsymbol{r_A \cdot r_B \cdot P}$
$\kappa = \text{Hash}(K)$
$tr = (s_B, E_B, s_A, E_A)$
$c_B = \text{HMAC}(\kappa, tr)$

Negotiate shared key. Similar to SPEKE (expired patent) but using a mask and scalar.

Dragonfly



Confirm peer negotiated same key

Multi-password support in WPA3

Can only have a **single "unbound" password**

- All other passwords are tied to a client's MAC address
  - Access Point (AP) can then use a matching (different) password
- In practice, we want to hand out many unbound passwords
  - Many users that don't connect in sequence, e.g., hotels or conference
- Bigger issue: clients may use MAC address randomization
  - Some randomize MAC address every day, even for the same network
  - We need a different solution…

Current multi-password solution in IEEE 802.11

- They introduced a **password identifier**
  - Essentially the same as a username
  - User must enter password identifier & password
  - Identifier sent in plaintext, Access Point (AP) uses matching password
- This has some drawbacks
  - User must remember and enter password & password identifier
  - Identifier is sent in plaintext, leaks info and enables user tracking

What does the industry seem to want?

- Solution where *only* a password needs to be entered
  - No ‘registration phase’. The password *is* the user identity!
- Passwords should be short just like with current WPA3
  - Don’t want to be entering longer passwords or extra information
- **Ideally same security guarantees** as single-password WPA3
- Avoid DoS attacks, in particular against the Access Point (AP)
  - In multi-password WPA2, the AP does for loop, so ideally not worse…
- *“Ideally minimal changes to Dragonfly to ease implementation”*
- *“Ideally support tens of **thousands of simultaneous passwords**”*

Naïve: do *n* parallel Dragonfly executions

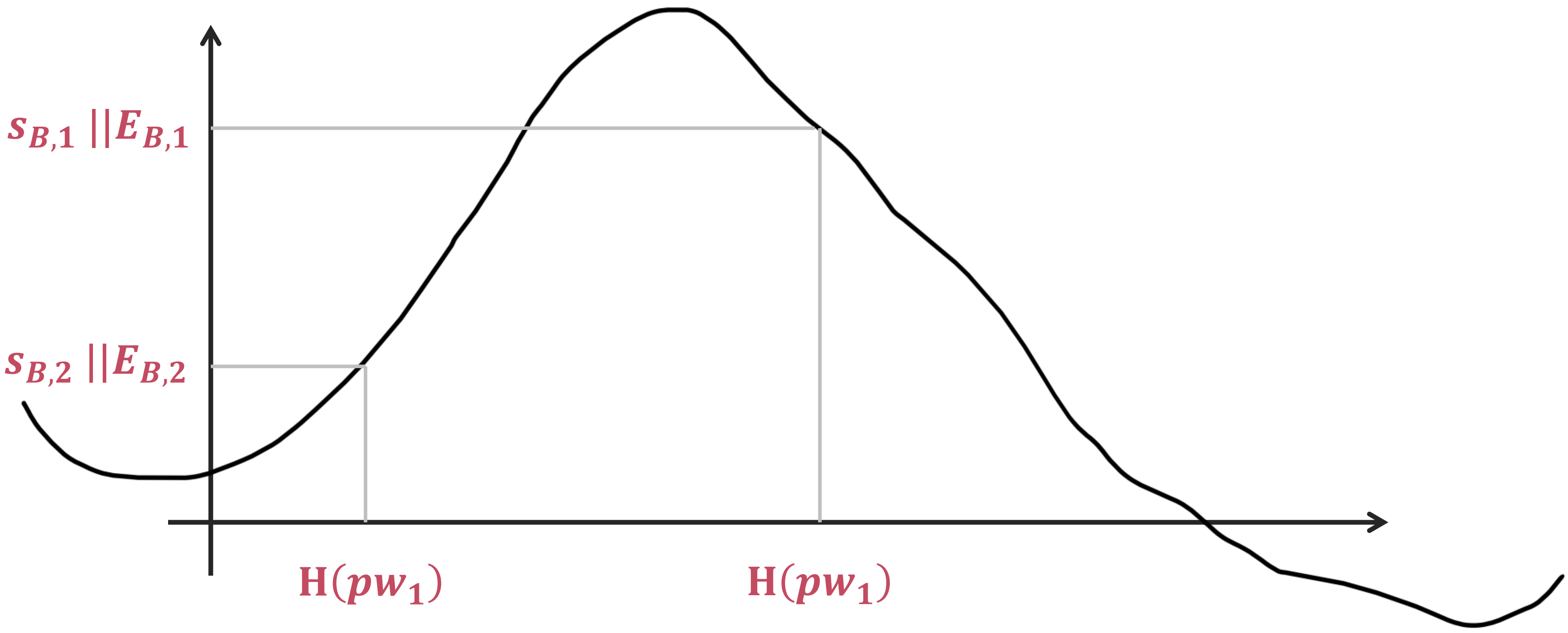
- Has obvious overhead:
  - All packets sent *n* times, all computations done *n* times
- We can do better: adapt O-PAKE or SweetPAKE [1,2]
- But a rogue AP can now **guess *n* passwords at once**!
  - General problem: reduces security compared to single-PW protocol. Unclear whether supporting that many passwords is a good idea?
  - Possible solution: client **waits for *n* seconds before reconnecting**
  - On average, online attack has same impact as single-PW protocol

Adapting O-PAKE [1]

O-PAKE can turn any PAKE into an *oblivious* PAKE

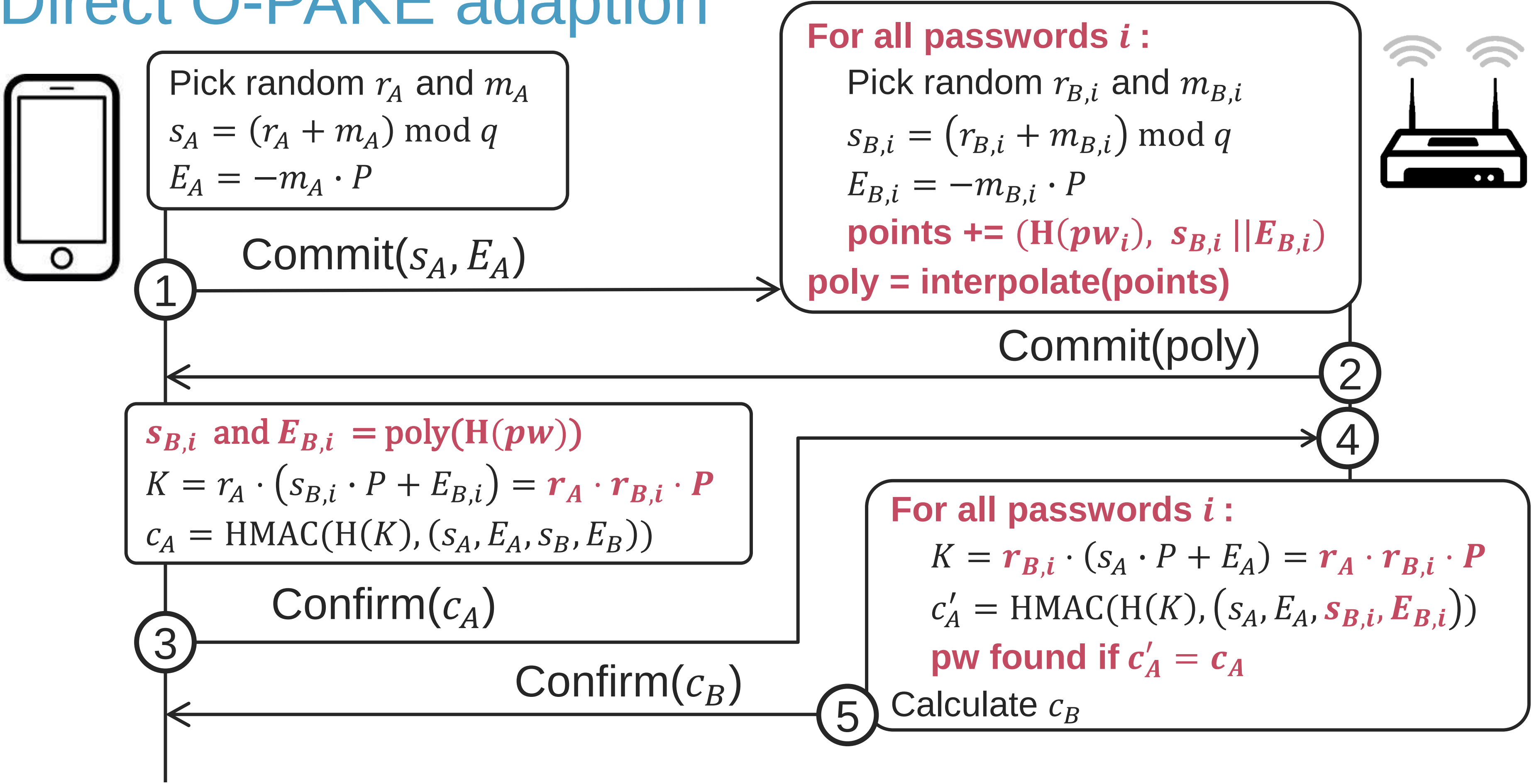
- Oblivious = client can try *n* passwords at once
- Based on Index-Hiding Message Encoding
  - Polynomial interpolation of points where:
  - X = hash(pw)
  - Y = encoded handshake message
- Polynomial coefficients are sent to the client
- Client recovers the right message by calculating f(hash(pw))

Polynomial interpolation idea



→ **Send poly coefficients** to the client. Client recovers $s_B\,||E_B$.

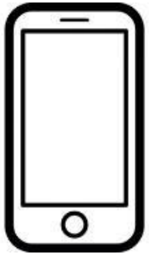
Direct O-PAKE adaption



Multi-Dragonfly

- Data overhead is O(c n) where n = #passwords
  - This seems hard to avoid…
  - …unless we can reuse data across handshakes?
  - …unless passwords are generated or have structure?
- First: can we **reduce the value of c in O(c n)**?
  - Reuse the same scalar for all passwords!
  - Note: what comes next are fresh ideas without any proofs…

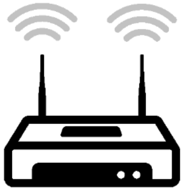
Direct O-PAKE adaption



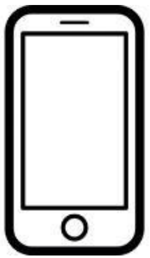
Pick random r_A and m_A
$s_A = (r_A + m_A) \bmod q$
$E_A = -m_A \cdot P$

1 Commit(s_A, E_A) →

For all passwords i :
Pick random $r_{B,i}$ and $m_{B,i}$
$s_{B,i} = (r_{B,i} + m_{B,i})\ mod\ q$
$E_{B,i} = -m_{B,i} \cdot P$
points += $(\mathrm{H}(pw_i),\ s_{B,i} || E_{B,i})$
poly = interpolate(points)



Reuse scalar



Pick random r_A and m_A

$s_A = (r_A + m_A) \bmod q$

$E_A = -m_A \cdot P$

1 Commit(s_A, E_A) →

Pick random s_B

For all passwords i :

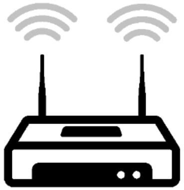
Pick random $r_{B,i}$ and $m_{B,i}$

$s_{B,i} = (r_{B,i} + m_{B,i})\ mod\ q$

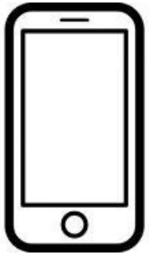
$E_{B,i} = -m_{B,i} \cdot P$

points += $(\mathrm{H}(pw_i),\ s_{B,i} \,||\, E_{B,i})$

poly = interpolate(points)



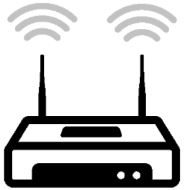
Reuse scalar



Pick random r_A and m_A
$s_A = (r_A + m_A) \bmod q$
$E_A = -m_A \cdot P$

1 Commit(s_A, E_A)

Pick random s_B
For all passwords i :
Pick random $r_{B,i}$ and $m_{B,i}$
$s_{B,i} = (r_{B,i} + m_{B,i}) \bmod q$
$E_{B,i} = -m_{B,i} \cdot P$
points += $(\mathrm{H}(pw_i),\ s_{B,i} \| E_{B,i})$
poly = interpolate(points)



Reuse scalar

Pick random r_A and m_A

$s_A = (r_A + m_A) \bmod q$

$E_A = -m_A \cdot P$

Commit(s_A, E_A) ①

Pick random s_B

For all passwords i :

Pick random $r_{B,i}$ and $m_{B,i}$

$\boldsymbol{r_{B,i} = (s_B - m_{B,i}) \bmod q}$

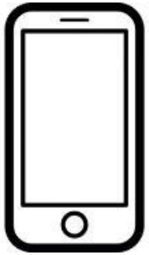
$E_{B,i} = -m_{B,i} \cdot P$

points += $(\mathrm{H}(pw_i),\ s_{B,i} \,||\, E_{B,i})$

poly = interpolate(points)

Commit(poly) ②

Reuse scalar



Pick random r_A and m_A

$s_A = (r_A + m_A) \bmod q$

$E_A = -m_A \cdot P$

① Commit(s_A, E_A) →

Pick random s_B

For all passwords i :

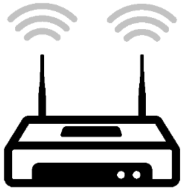
Pick random $r_{B,i}$ and $m_{B,i}$

$\boldsymbol{r_{B,i} = (s_B - m_{B,i})\ mod\ q}$

$E_{B,i} = -m_{B,i} \cdot P$

points += $(\mathrm{H}(pw_i),\ s_{B,i} \,||\, E_{B,i})$

poly = interpolate(points)



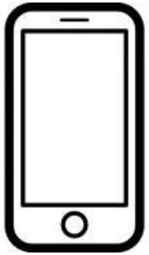
← Commit($\boldsymbol{s_B}$, poly) ②

$s_{B,i}$ and $E_{B,i} = \mathrm{poly}(\mathrm{H}(pw))$

$K = r_A \cdot (s_{B,i} \cdot P + E_{B,i}) = r_A \cdot r_{B,i} \cdot P$

$c_A = \mathrm{HMAC}(H(K), (s_A, E_A, s_B, E_B))$

Reuse scalar (final)

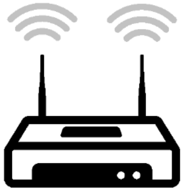


Pick random r_A and m_A

$s_A = (r_A + m_A) \bmod q$

$E_A = -m_A \cdot P$

① Commit(s_A, E_A) →



Pick random s_B

For all passwords i :

Pick random $r_{B,i}$ and $m_{B,i}$

$\boldsymbol{r_{B,i} = (s_B - m_{B,i}) \bmod q}$

$E_{B,i} = -m_{B,i} \cdot P$

points += $(\mathrm{H}(pw_i),\ s_{B,i} \| E_{B,i})$

poly = interpolate(points)

② ← Commit($\boldsymbol{s_B}$, poly)

$\boldsymbol{E_{B,i} = \mathrm{poly}(\mathrm{H}(pw))}$

$K = r_A \cdot (s_B \cdot P + E_{B,i}) = r_A \cdot r_{B,i} \cdot P$

$c_A = \mathrm{HMAC}(H(K), (s_A, E_A, s_B, E_B))$

③ Confirm(c_A) → ④

For all passwords i :

$K = r_{B,i} \cdot (s_A \cdot P + E_A) = r_A \cdot r_{B,i} \cdot P$

$c'_A = \mathrm{HMAC}(\mathrm{H}(K), (s_A, E_A, s_{B,i}, E_{B,i}))$

pw found if $c'_A = c_A$

Calculate c_B

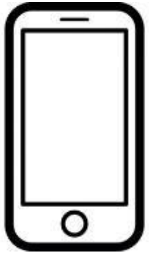
④ ← Confirm(c_B)

Multi-Dragonfly

- Data overhead is now lower!
- But still requires polynomial interpolation in every handshake
  - Can optimize with precomputation if passwords remain identical [3]
  - But still $O(n^2)$ in number of the passwords
- Do poly interpolation once and **reuse the polynomial**?
  - We can easily change the scalar s_B while keeping all $m_{B,i}$ the same
  - Would what this look like? Let's explore…

[3] M. Manulis and B. Poettering. Practical affiliation-hiding authentication from improved polynomial interpolation. In Asia CCS, 2011.

Reuse scalar (final)



Pick random r_A and m_A

$s_A = (r_A + m_A) \bmod q$

$E_A = -m_A \cdot P$

(1) Commit(s_A, E_A) →

Pick random s_B

For all passwords i :

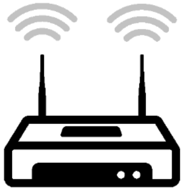
  Pick random $m_{B,i}$

  $r_{B,i} = (s_B - m_{B,i})\ mod\ q$

  $E_{B,i} = -m_{B,i} \cdot P$

  points += $(\mathrm{H}(pw_i), E_{B,i})$

poly = interpolate(points)



Reuse poly

Pick random r_A and m_A
$s_A = (r_A + m_A) \bmod q$
$E_A = -m_A \cdot P$

Pick random s_B
For all passwords i :
Pick random $m_{B,i}$
$\boldsymbol{r_{B,i} = (s_B - m_{B,i})\ mod\ q}$
$E_{B,i} = -m_{B,i} \cdot P$
points += $(\mathrm{H}(pw_i), E_{B,i})$
poly = interpolate(points)

① Commit(s_A, E_A) →

Reuse poly

Pick random r_A and m_A

$s_A = (r_A + m_A) \bmod q$

$E_A = -m_A \cdot P$

1 Commit(s_A, E_A) →

For all passwords i :

Pick random $m_{B,i}$

$E_{B,i} = -m_{B,i} \cdot P$

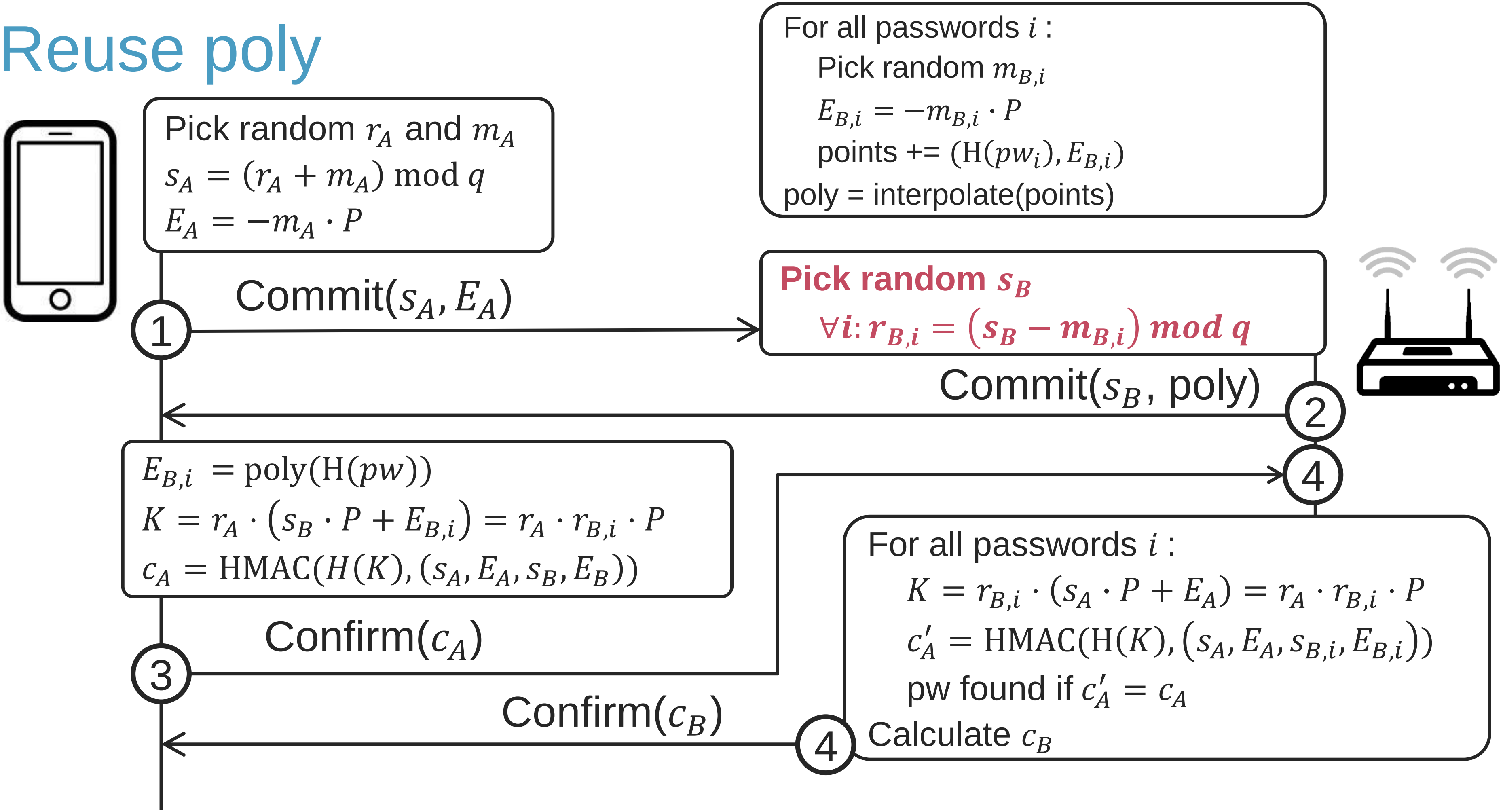
points += $(\mathrm{H}(pw_i), E_{B,i})$

poly = interpolate(points)

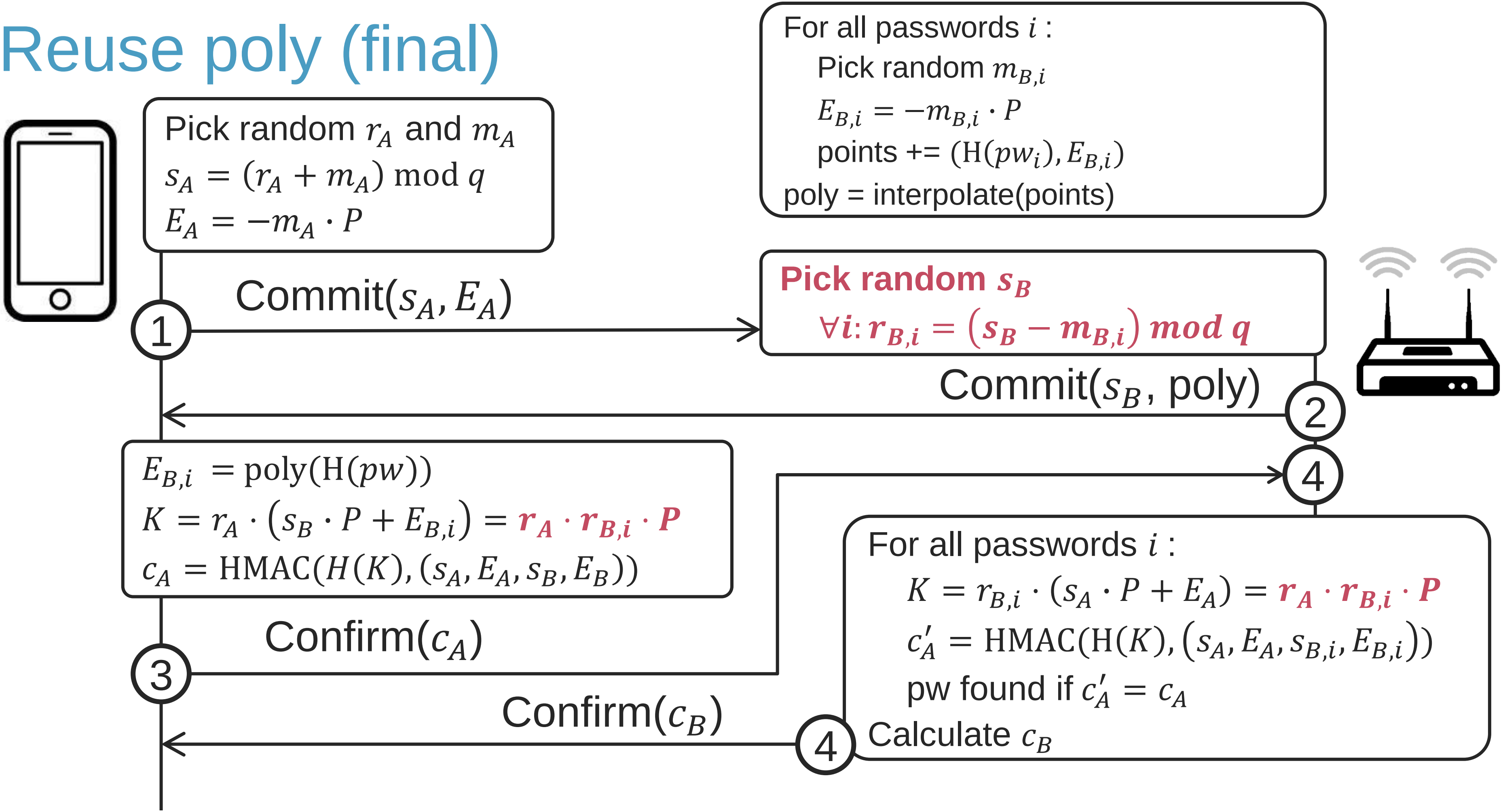
Pick random s_B

$\forall i: r_{B,i} = (s_B - m_{B,i}) \bmod q$

Reuse poly


For all passwords i :
Pick random $m_{B,i}$
$E_{B,i} = -m_{B,i} \cdot P$
points += $(\mathrm{H}(pw_i), E_{B,i})$
poly = interpolate(points)
Pick random r_A and m_A
$s_A = (r_A + m_A) \bmod q$
$E_A = -m_A \cdot P$
Commit(s_A, E_A)
1
Pick random s_B
$\forall i: r_{B,i} = (s_B - m_{B,i}) \bmod q$
Commit(s_B, poly)
2
$E_{B,i} = \mathrm{poly}(\mathrm{H}(pw))$
$K = r_A \cdot (s_B \cdot P + E_{B,i}) = r_A \cdot r_{B,i} \cdot P$
$c_A = \mathrm{HMAC}(H(K), (s_A, E_A, s_B, E_B))$
4
For all passwords i :
$K = r_{B,i} \cdot (s_A \cdot P + E_A) = r_A \cdot r_{B,i} \cdot P$
$c'_A = \mathrm{HMAC}(\mathrm{H}(K), (s_A, E_A, s_{B,i}, E_{B,i}))$
pw found if $c'_A = c_A$
Calculate c_B
Confirm(c_A)
3
Confirm(c_B)
4

Reuse poly (final)



Advantages

- Can broadcast the polynomial to all clients at once
  - Can even be sent outside the handshake…
  - …this makes supporting many passwords more feasible
- Reduces computational burden on the AP
  - AP still loops over all passwords, but so do existing WPA2 solutions

But is it secure?*

- Reuse of polynomial = reuse of first handshake message
  - **Doing so is secure for CPace [4].** So possibly also for Dragonfly?
  - CPace is similar to Dragonfly but more efficient…
- This seems to be the way forward to explore!
  - From academic perspective, we can continue with CPace
- Industry might be interested in updated proof of Dragonfly…
  - …the scalar s_B doesn't have to change, but it ensures fresh keys?
- **Help needed!** Eternal fame if WPA3 adopts your solution ☺
  - Does this look OK? Are new proofs needed? What about scalar s_B?

* Slide updated based on suggestions and insights after the presentation. Thank you!

Other directions

- Can also do similar things like SweetPAKE [2]
  - Based on Password-Authenticated Public-Key Encryption (PAPKE)
  - Not based on Dragonfly, IEEE 802.11 might be more hesitant to adopt
  - But also seems worth exploring!
- Could even combine polynomial interpolation with PAPKE
  - Happy to discuss, see backup slides
- Post-quantum? Currently not (yet) a focus in Wi-Fi…

Conclusion

- High interest to have multi-password WPA3 solutions

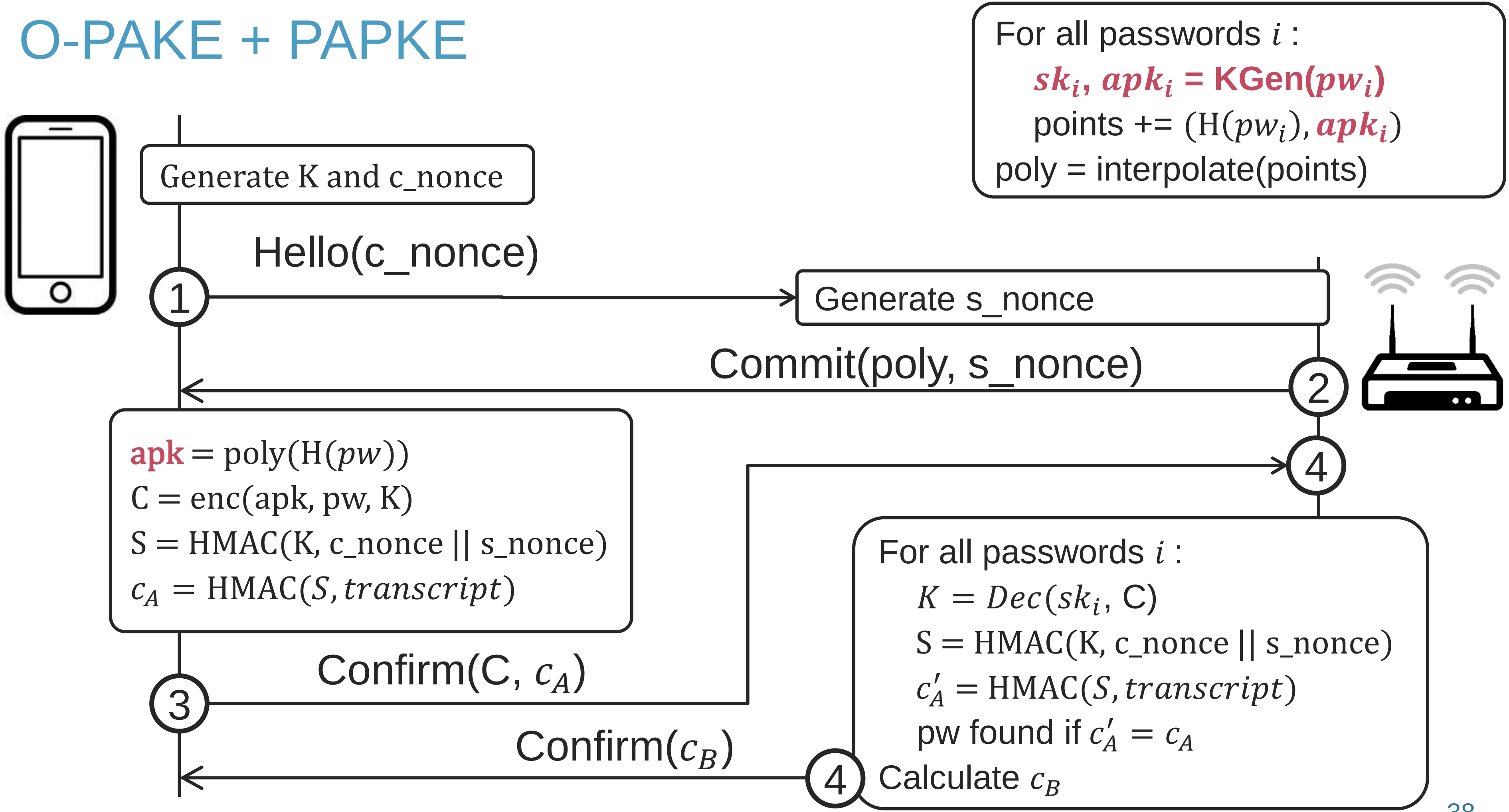
- Supporting low #password is feasible
- **Help needed to optimize solutions for more passwords!**
  - Security analysis, optimizations, ideas…
  - Eternal fame awaits! ☺

→ https://github.com/DistriNet/decoyauth

References

1. F. Kiefer and M. Manulis. Oblivious PAKE: Efficient handling of password trials. In Springer International Conference on Information Security, 2015.
2. A. Arriaga, P. Y. Ryan, and M. Skrobot. SweetPAKE: Key exchange with decoy passwords. In Asia CCS, 2024.
3. M. Manulis and B. Poettering. Practical affiliation-hiding authentication from improved polynomial interpolation. In Asia CCS, 2011.
4. D. Harkins. Simultaneous authentication of equals: A secure, password-based key exchange for mesh networks. In IEEE SensorComm, 2008.
5. Manuel Barbosa, Kai Gellert, Julia Hesse, and Stanislaw Jarecki. "Bare pake: universally composable key exchange from just passwords." In AICC Annual International Cryptology Conference, 2024.

O-PAKE + PAPKE



For all passwords i :
sk_i, apk_i = KGen(pw_i)
points += $(\mathrm{H}(pw_i), apk_i)$
poly = interpolate(points)
Generate K and c_nonce
Hello(c_nonce)
1
Generate s_nonce
Commit(poly, s_nonce)
2
$\text{apk} = \text{poly}(\mathrm{H}(pw))$
C = enc(apk, pw, K)
S = HMAC(K, c_nonce || s_nonce)
$c_A = \mathrm{HMAC}(S, transcript)$
4
Confirm(C, c_A)
3
Confirm(c_B)
4
For all passwords i :
$K = Dec(sk_i$, C)
S = HMAC(K, c_nonce || s_nonce)
$c'_A = \mathrm{HMAC}(S, transcript)$
pw found if $c'_A = c_A$
Calculate c_B

O-PAKE + PAPKE

- Included client and server nonce (c_nonce and s_nonce) to allow reuse of the polynomial while preventing replays
  - Client nonce likely not needed, since it already generates the key K
- Unclear how the data & computation overhead compares to other solutions. How expensive it PAPKE?
- Deviations more from Dragonfly, Wi-Fi vendors and/or IEEE 802.11 might be more hesitant to adopt it?

Scaling to *thousands* of passwords?!

Have different types of passwords

- **Unbounded** passwords: can be used by any client
  - After first usage, they are bound to the client's MAC address
- **Bound** passwords: associated to a client's MAC address
- **Group** passwords: can always be used by any client
  - Never get bound to a specific MAC address

Need to support fewer actual simultaneous passwords!

- Trickier nowadays due to MAC address randomization

Use password identifier in the background

Use password identifier instead of MAC address

Proposal to regularly **rotate the password identifier**

- After connecting, network issues a (new) password identifier
- Must synchronize identifier across all devices that use a particular password
  - Standard currently does not specify how to do this
  - But it does look feasible with some effort