DecoyAuth:

Authentication with Compromise Detection

Mathy Vanhoef

February 2025

Funded by NGI Sargasso under the DecoyAuth project.

Stolen credentials still a big issue

Verizon 2024 data breach report (30,458 security incidents)

- Stolen credentials still the top cause of breaches

Solution: decoy tokens

- Act as reverse honeypot
- Use of a decoy token means a breach occurred

Problem

Zero-Knowledge Authentication (ZK-Auth)

- Counterparty *only* learns if token was correct
- Downside: **can’t use decoy tokens**!

We **add support of decoy tokens** to ZK protocols

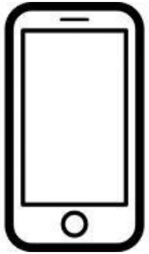
- Decoy token is indistinguishable from a real token
- If decoy token is used: take appropriate measures

Objective: next-gen security and identity

1. Design the DecoyAuth protocol.
   - Based on Dragonfly.
2. Make a reference implementation
   - Will be open-sourced. Integrate into EAP authentication framework.
3. Create an open specification

Open standardization is core goal

Dragonfly



Pick random r_A and m_A

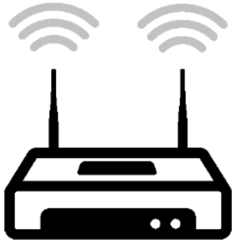
$s_A = (r_A + m_A) \bmod q$

$E_A = -m_A \cdot P$

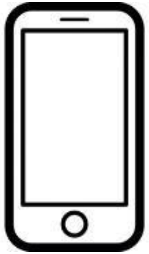
Pick random r_B and m_B

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

$E_B = -m_B \cdot P$



Dragonfly



Pick random r_A and m_A

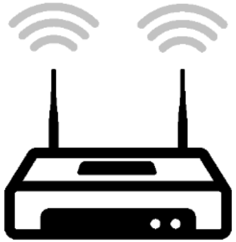
$s_A = (r_A + m_A) \bmod q$

$E_A = -m_A \boxed{P}$

Pick random r_B and m_B

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

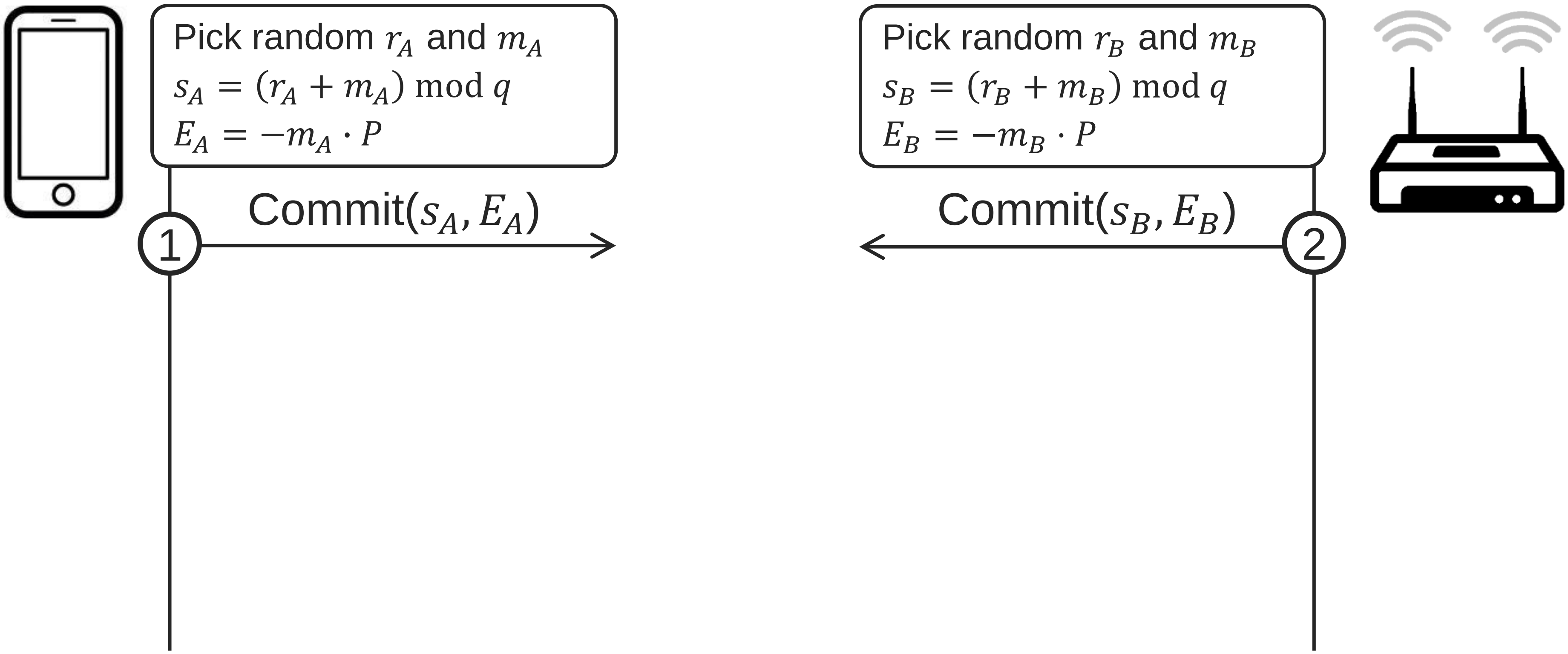
$E_B = -m_B \boxed{P}$



Password is hashed to group element P

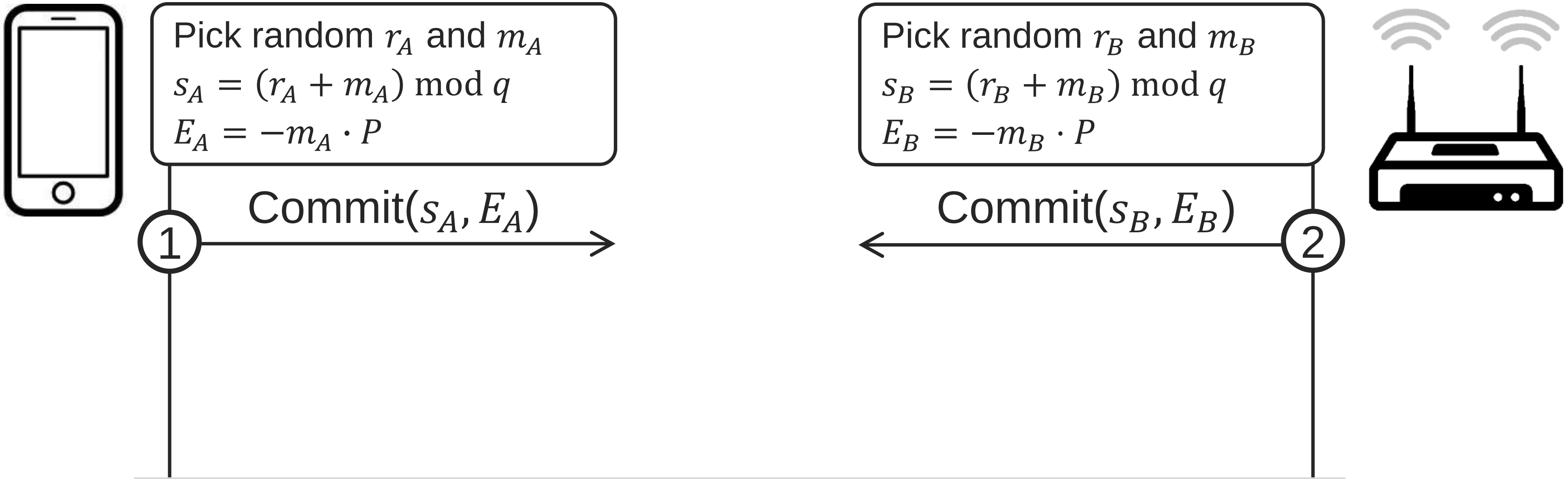
(Simplified Shallue Woestijne-Ulas)

Dragonfly



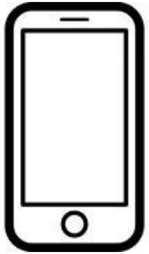
Pick random r_A and m_A
$s_A = (r_A + m_A) \bmod q$
$E_A = -m_A \cdot P$
Commit(s_A, E_A)
1
Pick random r_B and m_B
$s_B = (r_B + m_B) \bmod q$
$E_B = -m_B \cdot P$
Commit(s_B, E_B)
2

Dragonfly



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

Dragonfly



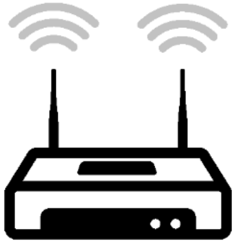
Pick random r_A and m_A

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

$E_A = -m_A \cdot P$

Commit(s_A, E_A)

1



Pick random r_B and m_B

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

$E_B = -m_B \cdot P$

Commit(s_B, E_B)

2

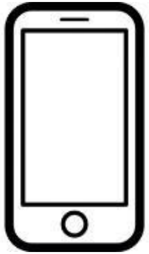
$$\begin{aligned} K &= r_A \cdot (s_B \cdot P + E_B) \\ &= r_A \cdot (r_B \cdot P + m_B \cdot P - m_B \cdot P) \\ &= r_A \cdot r_B \cdot P \end{aligned}$$

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

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

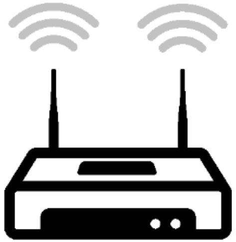
$c_A = \mathrm{HMAC}(\kappa, tr)$

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

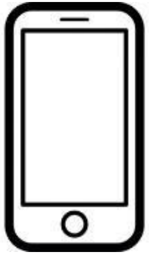


1 Commit(s_A, E_A) →

← Commit(s_B, E_B) 2

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

Dragonfly



Pick random r_A and m_A

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

$E_A = -m_A \cdot P$

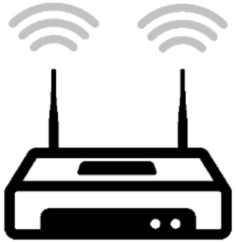
1 Commit(s_A, E_A) $\rightarrow$

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

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

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

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



Pick random r_B and m_B

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

$E_B = -m_B \cdot P$

2 $\leftarrow$ Commit(s_B, E_B)

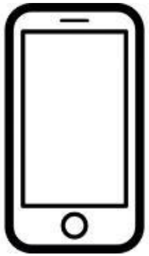
$K = r_B \cdot (s_A \cdot P + E_A) = \boldsymbol{r_A \cdot r_B \cdot P}$

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

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

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

Dragonfly



Pick random r_A and m_A

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

$E_A = -m_A \cdot P$

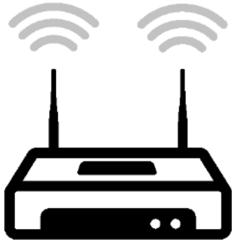
① $\text{Commit}(s_A, E_A) \rightarrow$

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

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

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

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



Pick random r_B and m_B

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

$E_B = -m_B \cdot P$

② $\leftarrow \text{Commit}(s_B, E_B)$

$K = r_B \cdot (s_A \cdot P + E_A) = \boldsymbol{r_A \cdot r_B \cdot P}$

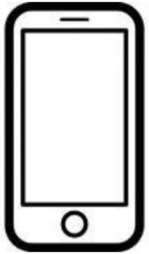
$\kappa = \text{Hash}(K)$

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

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

Negotiate shared key. Similar to SPEKE (expired patent) but using a <u>m</u>ask and <u>s</u>calar.

Dragonfly

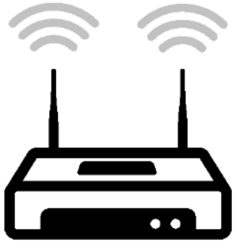


Pick random r_A and m_A

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

$E_A = -m_A \cdot P$

1 Commit(s_A, E_A) →



Pick random r_B and m_B

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

$E_B = -m_B \cdot P$

← Commit(s_B, E_B) 2

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

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

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

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

$K = r_B \cdot (s_A \cdot P + E_A) = \boldsymbol{r_A \cdot r_B \cdot P}$

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

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

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

3 Confirm(c_A) →

← Confirm(c_B) 4

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) →

2 ← Commit(s_B, E_B)

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

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

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

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

$K = r_B \cdot (s_A \cdot P + E_A) = \boldsymbol{r_A \cdot r_B \cdot P}$

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

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

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

3 Confirm(c_A) →

4 ← Confirm(c_B)

Confirm peer negotiated same key

What do people seem to want?

- Solution should support any key type including passwords
- **Ideally same security guarantees** as normal Dragonfly
- Avoid DoS attacks, in particular against the server
- *“Ideally minimal changes to Dragonfly to ease implementation”*
- *“Ideally support tens of **thousands of decoy keys**”*

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]

Adapting O-PAKE

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

- Oblivious = client can try *n* keys at once
- Authentication succeeds if any of out *n* keys is valid
- Server sends encoding of points to the client
- Client recovers the right message using its key

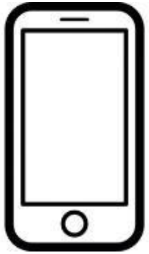
Adapting O-PAKE

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

- Oblivious = client can try *n* keys at once
- Authentication succeeds if any of out *n* keys is valid
- Server sends encoding of points to the client
- Client recovers the right message using its key

→ We reverse direction & apply to Dragonfly

Direct O-PAKE adaption

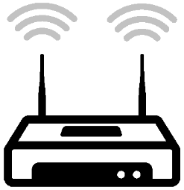


Pick random r_A and m_A

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

$E_A = -m_A \cdot P$

1 Commit(s_A, E_A)



For all passwords i :

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

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

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

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

vals = encode(points)

2 Commit(vals)

$s_{B,i}$ and $E_{B,i}$ = decode(vals, H(pw))

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

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

3 Confirm(c_A)

4

For all passwords i :

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

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

pw found if $c'_A = c_A$

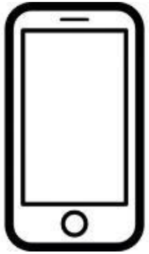
Calculate c_B

5 Confirm(c_B)

Decoy-Dragonfly

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

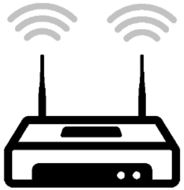
Direct O-PAKE adaption



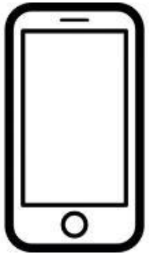
Pick random r_A and m_A
$s_A = (r_A + m_A) \bmod q$
$E_A = -m_A \cdot P$

1 Commit(s_A, E_A)

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

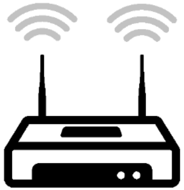
Pick random $r_{B,i}$ and $m_{B,i}$

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

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

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

vals = encode(points)



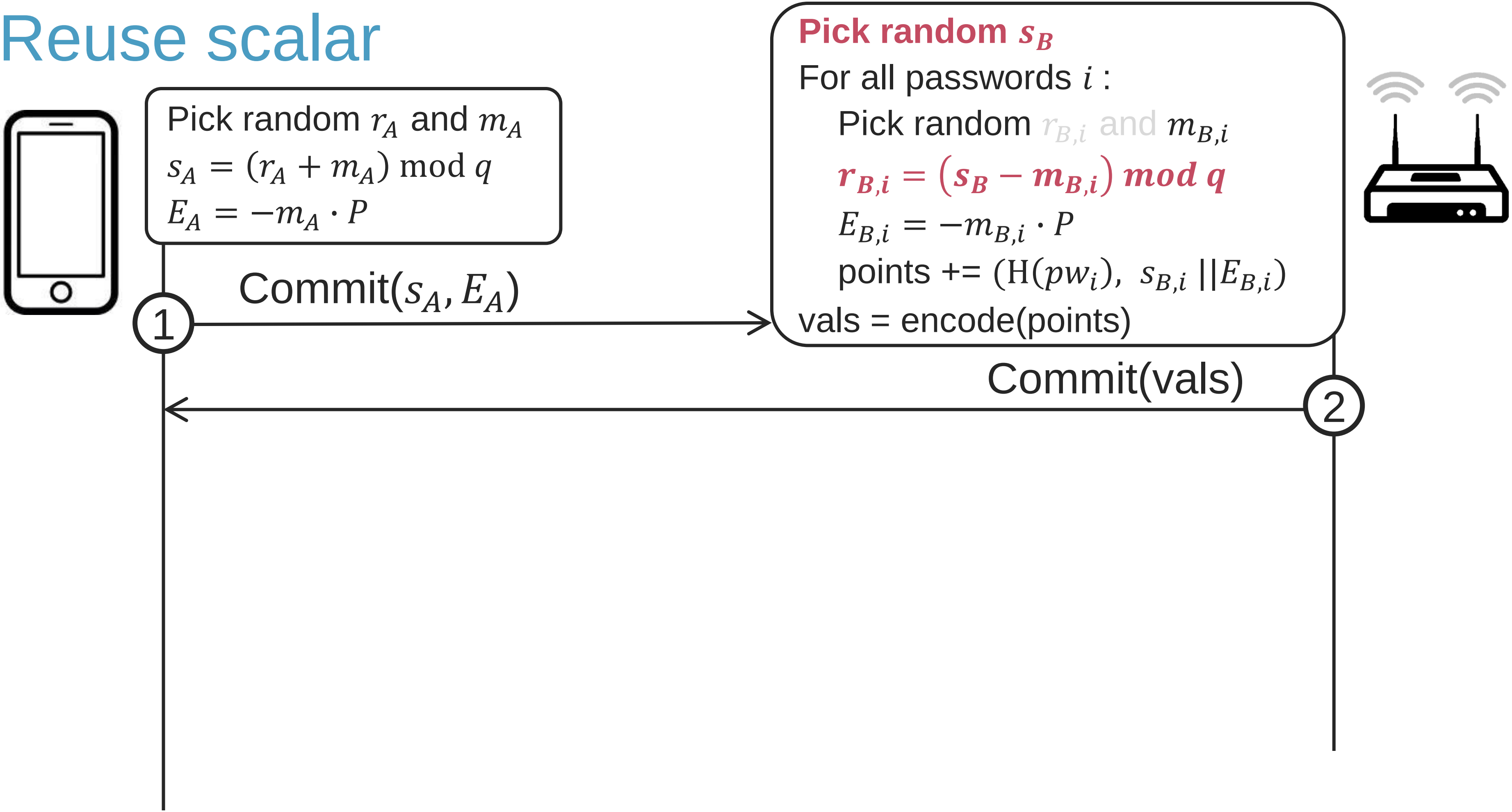
Reuse scalar

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

Reuse scalar



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

vals = encode(points)

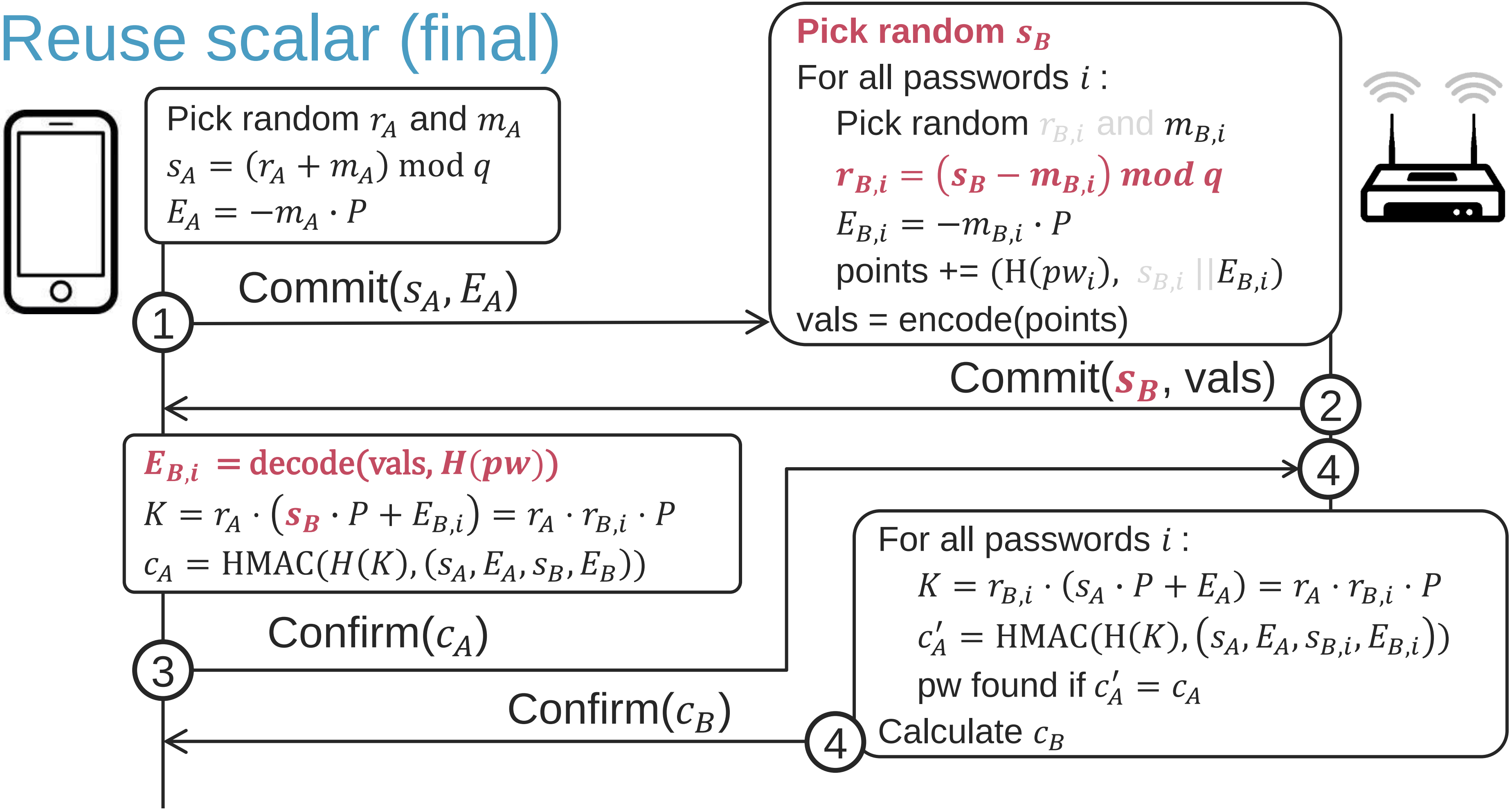
Commit$(\boldsymbol{s_B}$, vals) ②

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

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

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

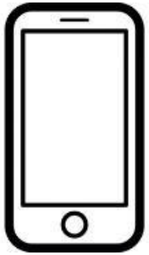
Reuse scalar (final)



Decoy-Dragonfly

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

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 :

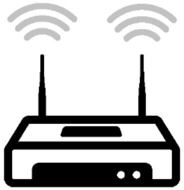
Pick random $m_{B,i}$

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

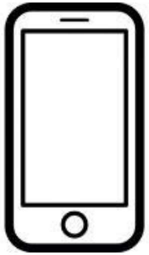
$E_{B,i} = -m_{B,i} \cdot P$

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

vals = encode(points)



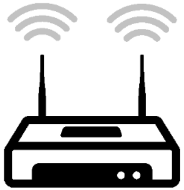
Reuse values



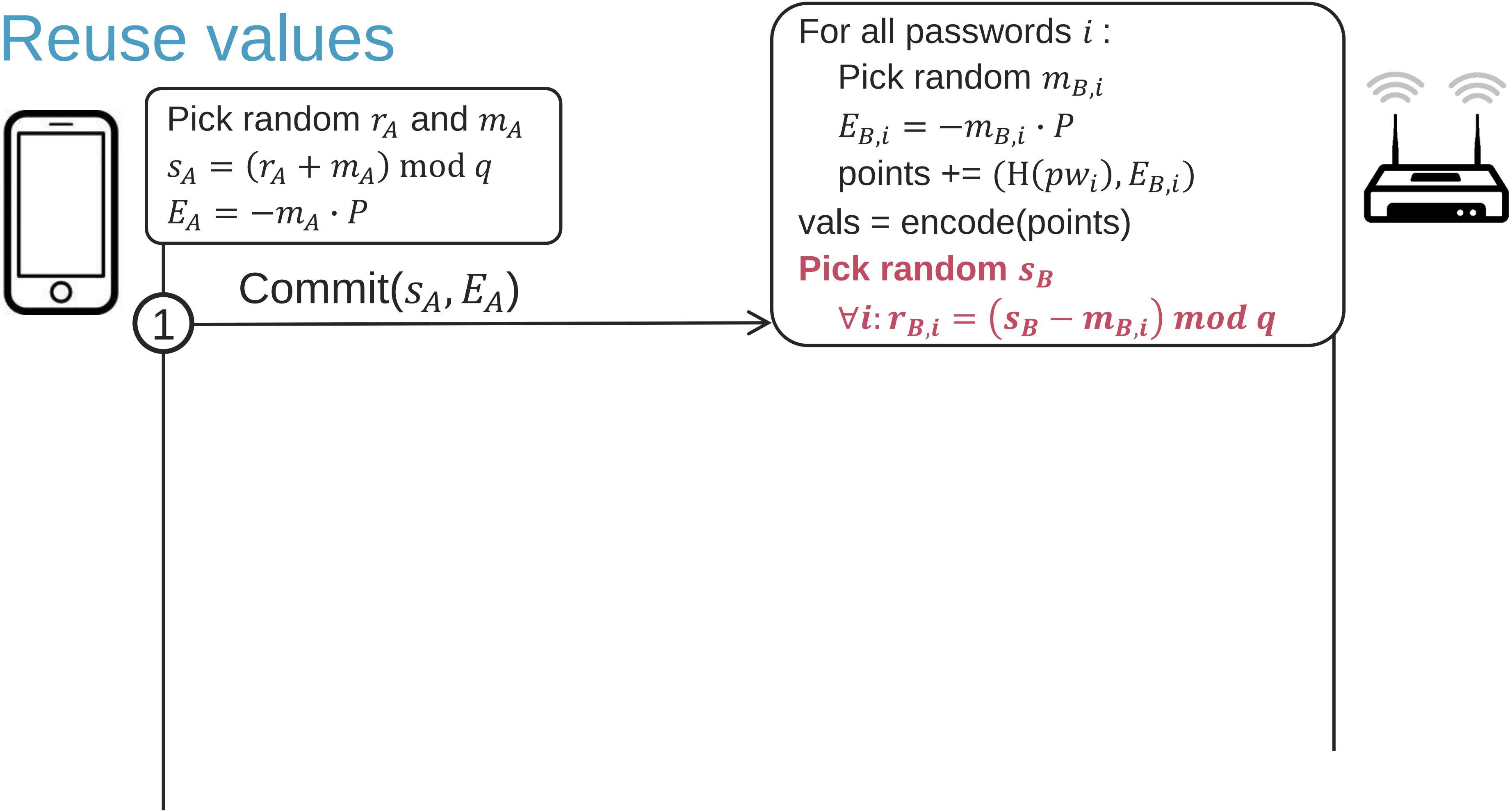
Pick random r_A and m_A
$s_A = (r_A + m_A) \bmod q$
$E_A = -m_A \cdot P$

1 Commit(s_A, E_A)

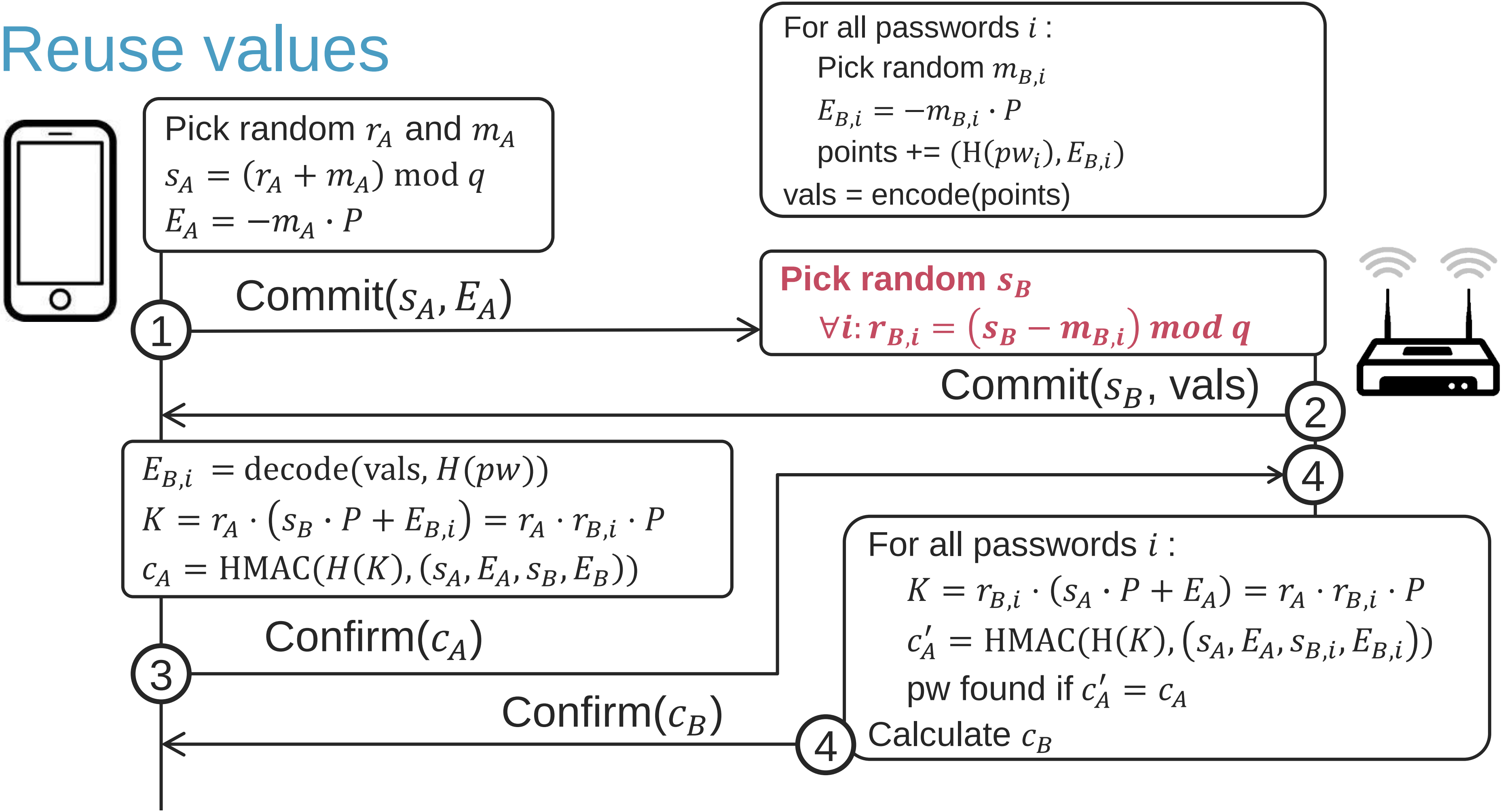
Pick random s_B
For all passwords i :
Pick random $m_{B,i}$
$\boldsymbol{r_{B,i} = (s_B - m_{B,i})\ mod\ q}$
$E_{B,i} = -m_{B,i} \cdot P$
points += $(\mathrm{H}(pw_i), E_{B,i})$
vals = encode(points)



Reuse values



Reuse values



Reuse values (final)

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

For all passwords i :
Pick random $m_{B,i}$
$E_{B,i} = -m_{B,i} \cdot P$
points += $(\mathrm{H}(pw_i), E_{B,i})$
vals = encode(points)

1 Commit(s_A, E_A)

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

2 Commit$(s_B$, vals$)$

$E_{B,i} = \mathrm{decode}(\mathrm{vals}, H(pw))$
$K = r_A \cdot (s_B \cdot P + E_{B,i}) = \boldsymbol{r_A \cdot r_{B,i} \cdot P}$
$c_A = \mathrm{HMAC}(H(K), (s_A, E_A, s_B, E_B))$

3 Confirm(c_A)

4

For all passwords i :
$K = r_{B,i} \cdot (s_A \cdot P + E_A) = \boldsymbol{r_A \cdot r_{B,i} \cdot P}$
$c'_A = \mathrm{HMAC}(\mathrm{H}(K), (s_A, E_A, s_{B,i}, E_{B,i}))$
pw found if $c'_A = c_A$
Calculate c_B

4 Confirm(c_B)

Advantages

- Can broadcast the encoded values to all clients at once
  - Can even be sent outside the handshake…
  - …this makes supporting many keys more feasible
- Reduces computational burden on the AP
  - AP still loops over all keys, but seems hard to avoid

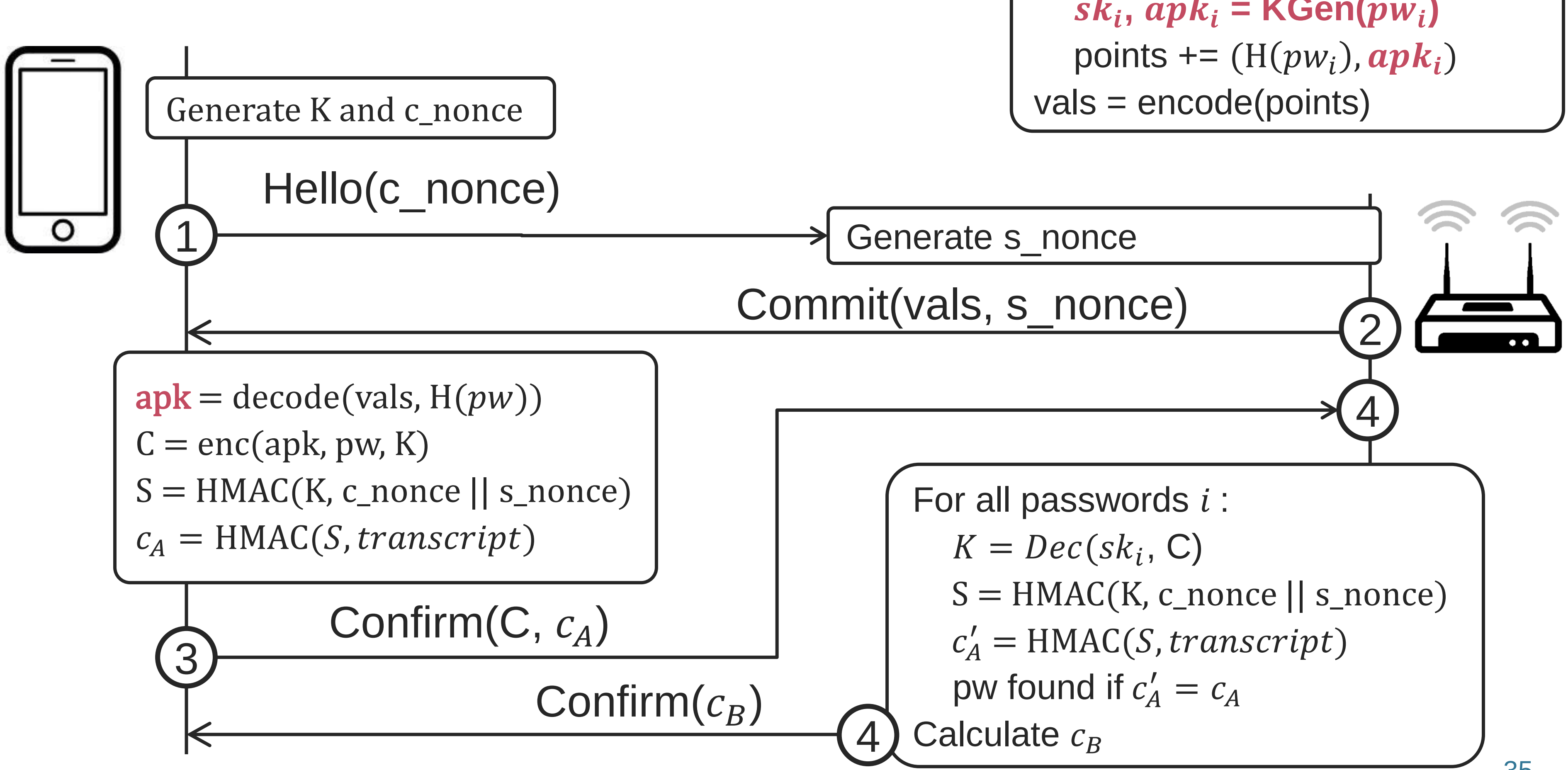
Other directions

- 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 point encoding with PAPKE
  - Happy to discuss, see backup slides

- Post-quantum? Currently not (yet) a focus in Wi-Fi…

O-PAKE + PAPKE



For all passwords i :
sk_i, apk_i = KGen(pw_i)
points += $(\mathrm{H}(pw_i), apk_i)$
vals = encode(points)
Generate K and c_nonce
Hello(c_nonce)
1
Generate s_nonce
Commit(vals, s_nonce)
2
apk = decode(vals, H(pw))
C = enc(apk, pw, K)
S = HMAC(K, c_nonce || s_nonce)
$c_A = \mathrm{HMAC}(S, transcript)$
Confirm(C, c_A)
3
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
Confirm(c_B)
4

Future extensions

Multi-password Wi-Fi feature

- Implemented by practically **all vendors for WPA2!**

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

- No longer possible with WPA3, because it uses Dragonfly…
- …we are looking into adding this feature as well

Advantage of multi-password WPA3

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

Conclusion

- Supporting decoy keys is feasible
- **Help needed to optimize solutions!**
  - 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. D. Harkins. Simultaneous authentication of equals: A secure, password-based key exchange for mesh networks. In IEEE SensorComm, 2008.