Efficient Asymmetric PAKE Compiler from KEM and AE

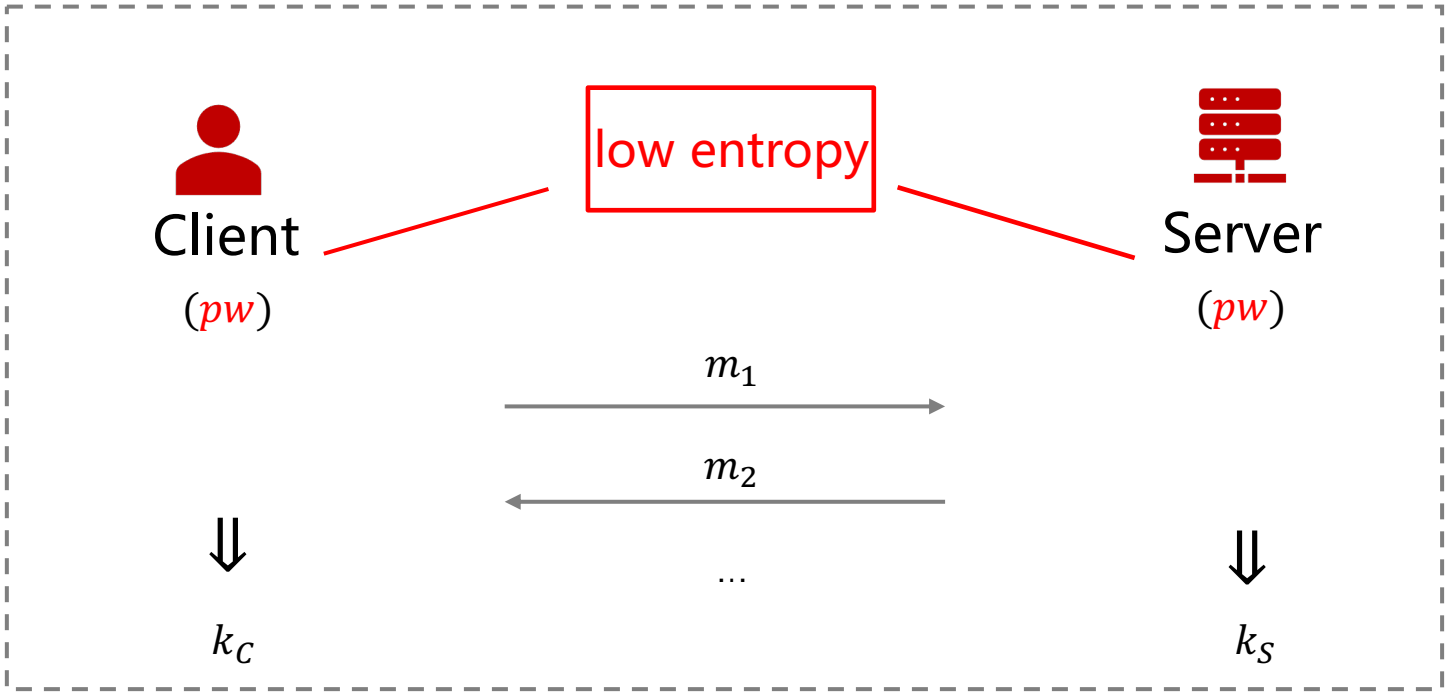
You Lyu, Shengli Liu, Shuai Han
Shanghai Jiao Tong University

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Contents

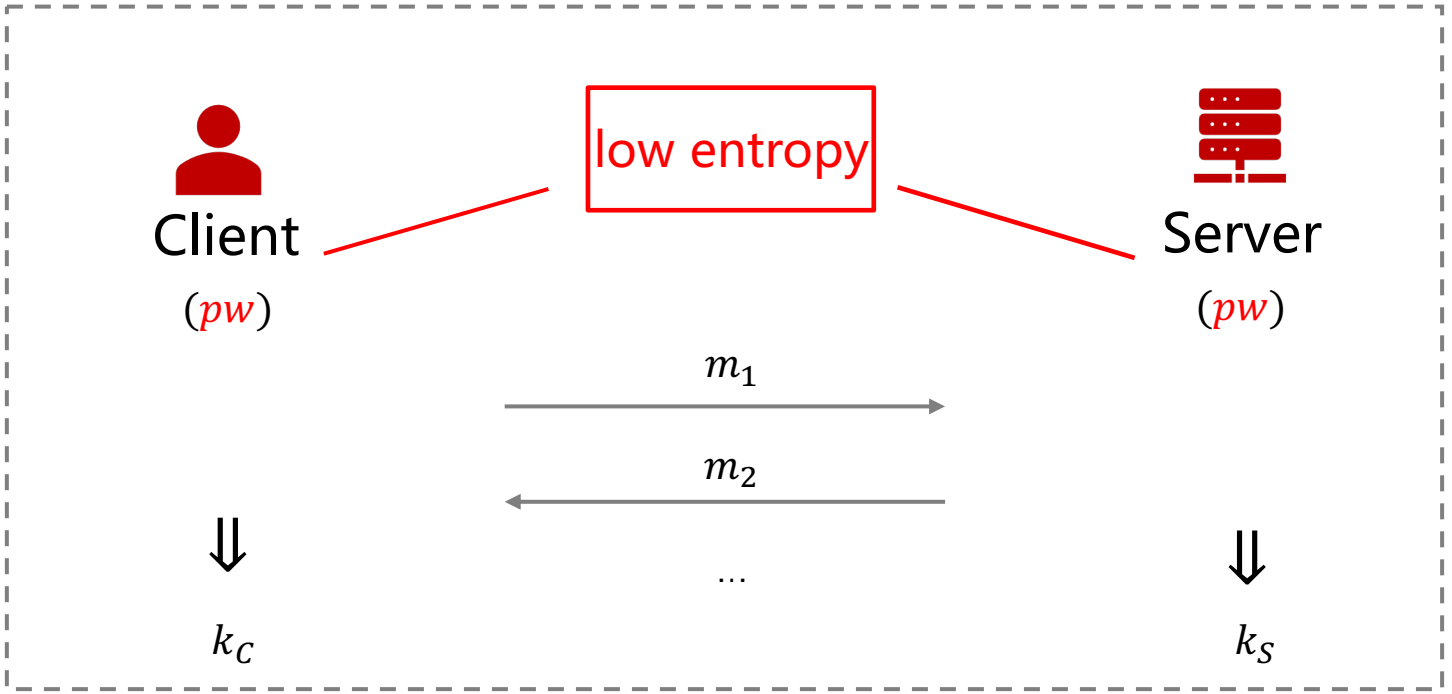
Contents

Password Authenticated Key Exchange



The Goal of PAKE： Client and Server can exchange a pseudo-random session key.

Password Authenticated Key Exchange

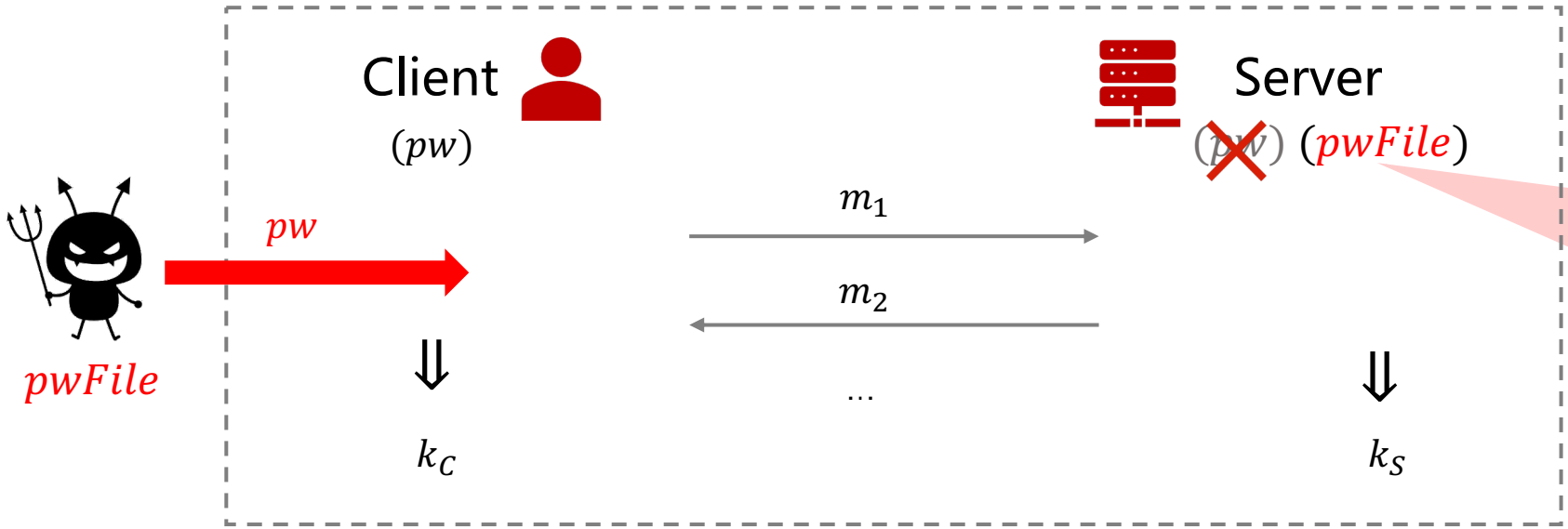


Note that the password only has low entropy.

Security Requirements:

- The best strategy for adversary is to implement online-dictionary attack (guess password online)
- Resist offline-dictionary attack

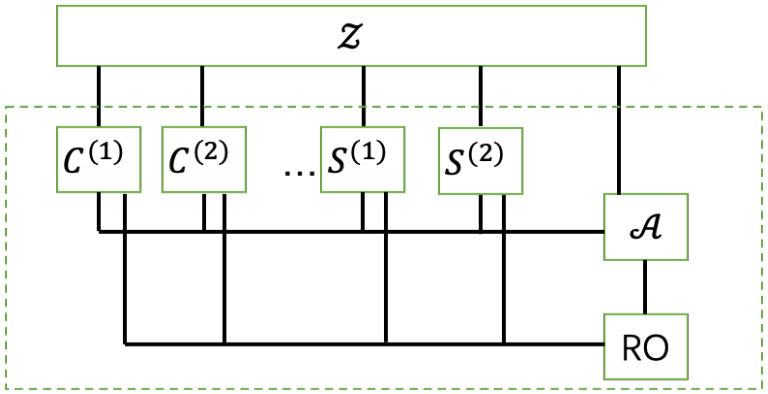
Asymmetric PAKE: Resist Server Compromise



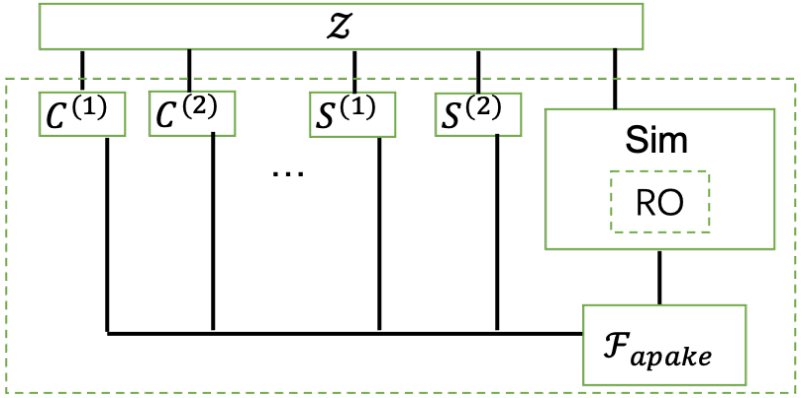
Security Requirements:

- The best strategy for adversary is to implement online-dictionary attack
- $\mathcal{A}$ can compute or pre-compute $[pw', F(pw')]$ pairs, and check whether $pwFile = F(pw')$
- $\mathcal{A}$ cannot impersonate a client unless it finds pw s.t. $F(pw) = pwFile$ via the above brute-force search

UC-security for aPAKE



Real World

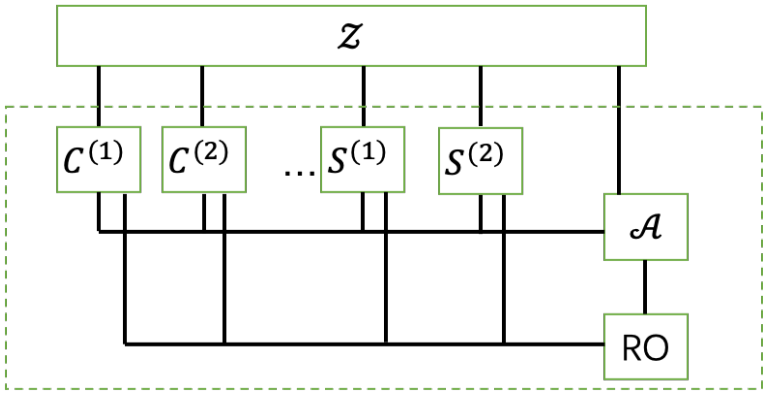


Ideal World

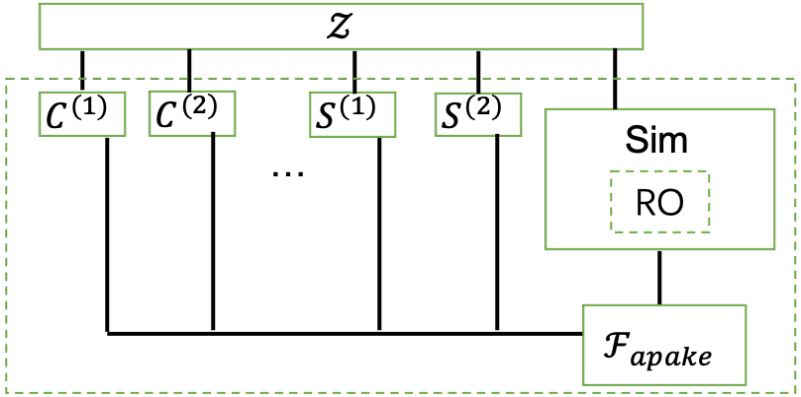
Roughly speaking, to prove UC security, we need to construct a simulator Sim s.t.

- Sim can simulate **indistinguishable transcript** of aPAKE protocol in the real world.
- Sim can simulate **indistinguishable output session key** for each client/server.

UC-security for aPAKE



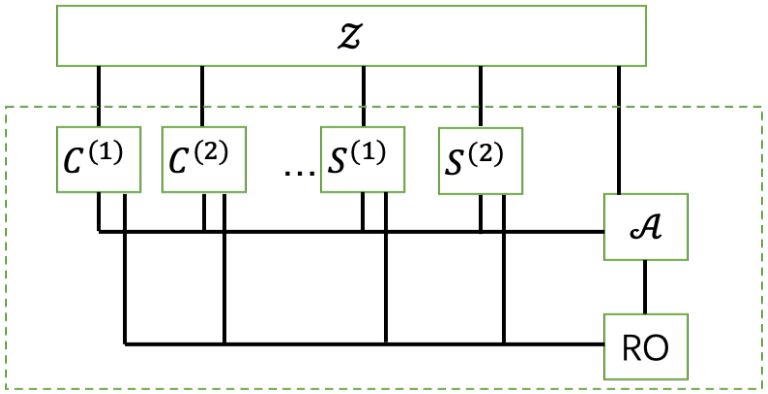
Real World



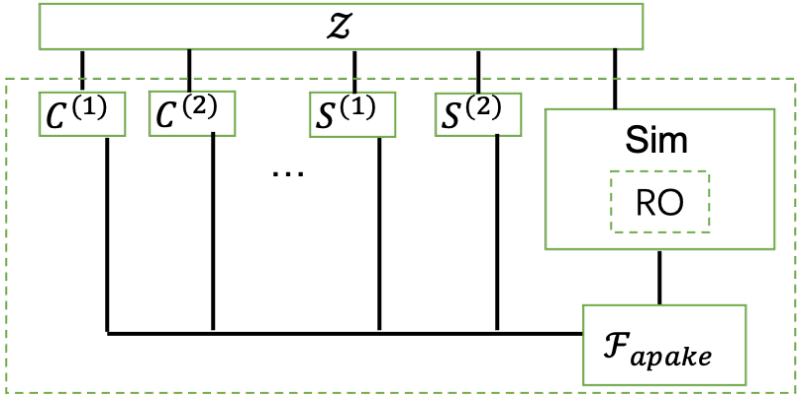
Ideal World

- Roughly speaking, to prove UC security, we need to construct a simulator Sim s.t.
  - Sim can simulate **indistinguishable transcript** of aPAKE protocol in the real world.
  - Sim can simulate **indistinguishable output session key** for each client/server.
- Sim has no information of password, except
  - A Testpw() oracle: model the online password-guessing attack — once for a client/server instance
  - An OfflineTestpw() oracle: model the offline brute-force search attack — at most q times in total

UC-security for aPAKE



Real World



Ideal World

UC-security is preferable!

UC-security **outperforms** IND-security

pw distribution:	**arbitrary pw** distributions ✓	**uniformly** distributed pw over a dictionary
composability:	Free secure composition ✓	**Composition security is not guaranteed**

aPAKE From PQ Assumptions

➢ Related works on PQ aPAKEs

Compiler	Building Block	Limitation
Ω-method [GMR06]	PAKE + signature	Lattice-based signature is not as efficient as KEM
OPAQUE[JKX18]	OPRF + AKE	No lattice-based UC-secure OPRF instantiations
[GJK21, SGJS22]	Key-Hiding AKE + ideal cipher	No lattice-based Key-Hiding AKE exists
[MX23]	PAKE + group action	No lattice-based instantiations

➢ Our Question:

Can we design an efficient aPAKE scheme instantiable from lattices?

Our Contribution

- A New Generic Compiler for UC-secure aPAKE in ROM:

  UC-secure aPAKE $\Leftarrow$ UC-secure PAKE + OW-PCA secure KEM + one-time secure AE

- Our compiler enjoys the following advantages:

  a) Efficient aPAKE scheme from lattices

  b) Mutual explicit authentication

  c) Good round efficiency

  d) Tight aPAKE scheme if the underlying PAKE and KEM has tight security

Contents

aPAKE compiler from KEM and AE

Registration: $pwFile = (rw, pk)$, where $rw||r = H(pw)$, $(pk, sk) \leftarrow \mathrm{KEM.KeyGen}(r)$

Client (pw)

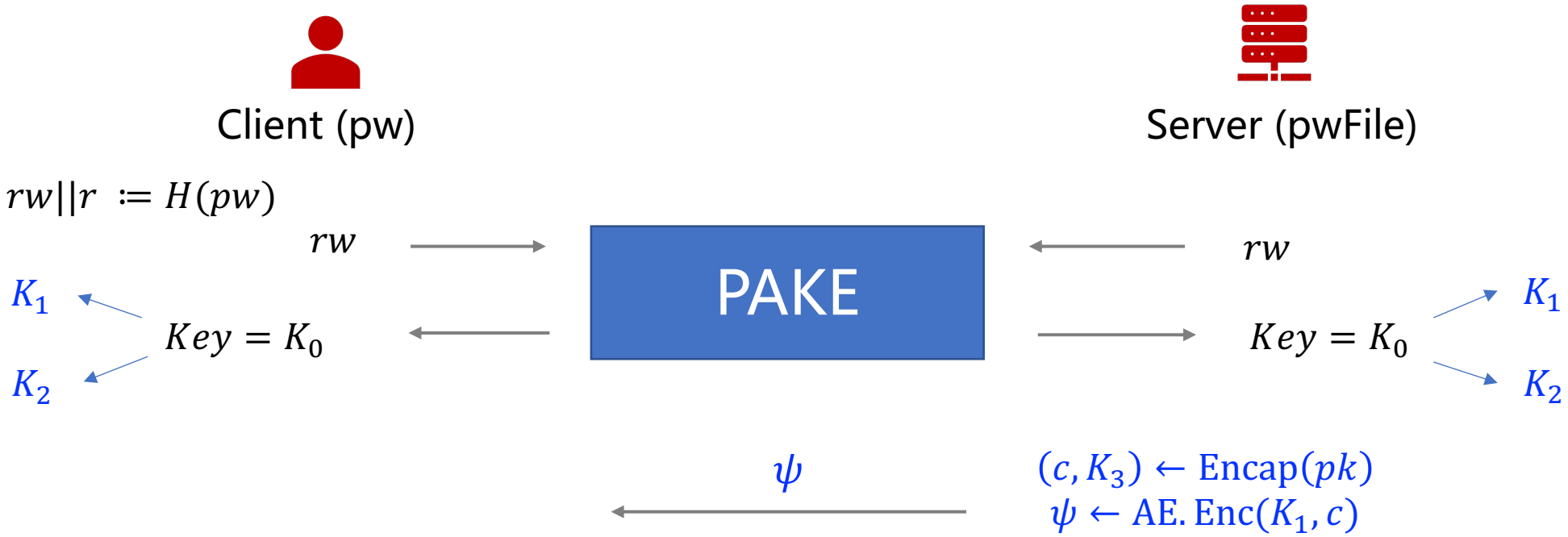
Server (pwFile)

aPAKE compiler from KEM and AE

Registration: $pwFile = (rw, pk)$, where $rw||r = H(pw)$, $(pk, sk) \leftarrow \mathrm{KEM.KeyGen}(r)$

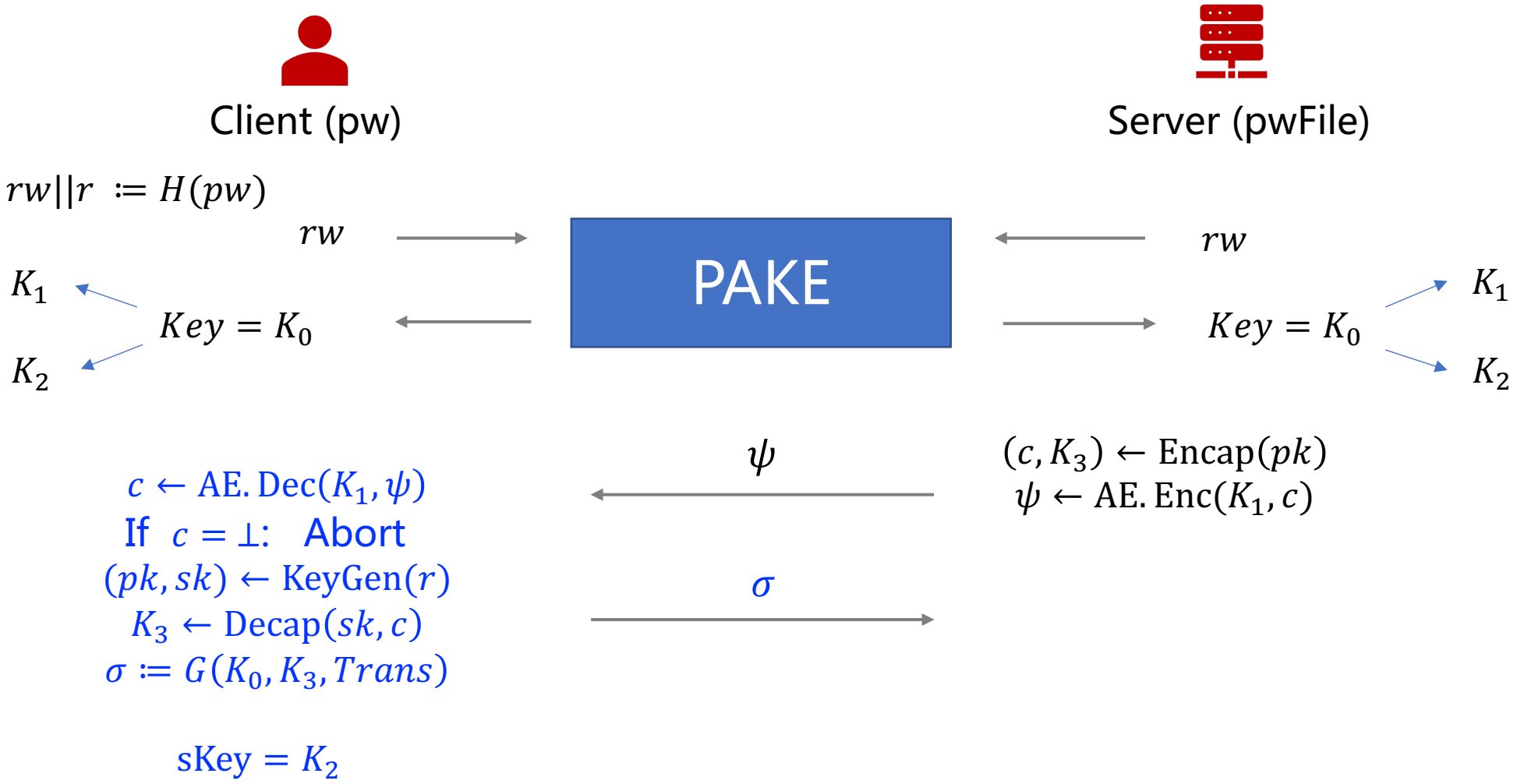
aPAKE compiler from KEM and AE

Registration: $pwFile = (rw, pk)$, where $rw||r = H(pw)$, $(pk, sk) \leftarrow \text{KEM.KeyGen}(r)$



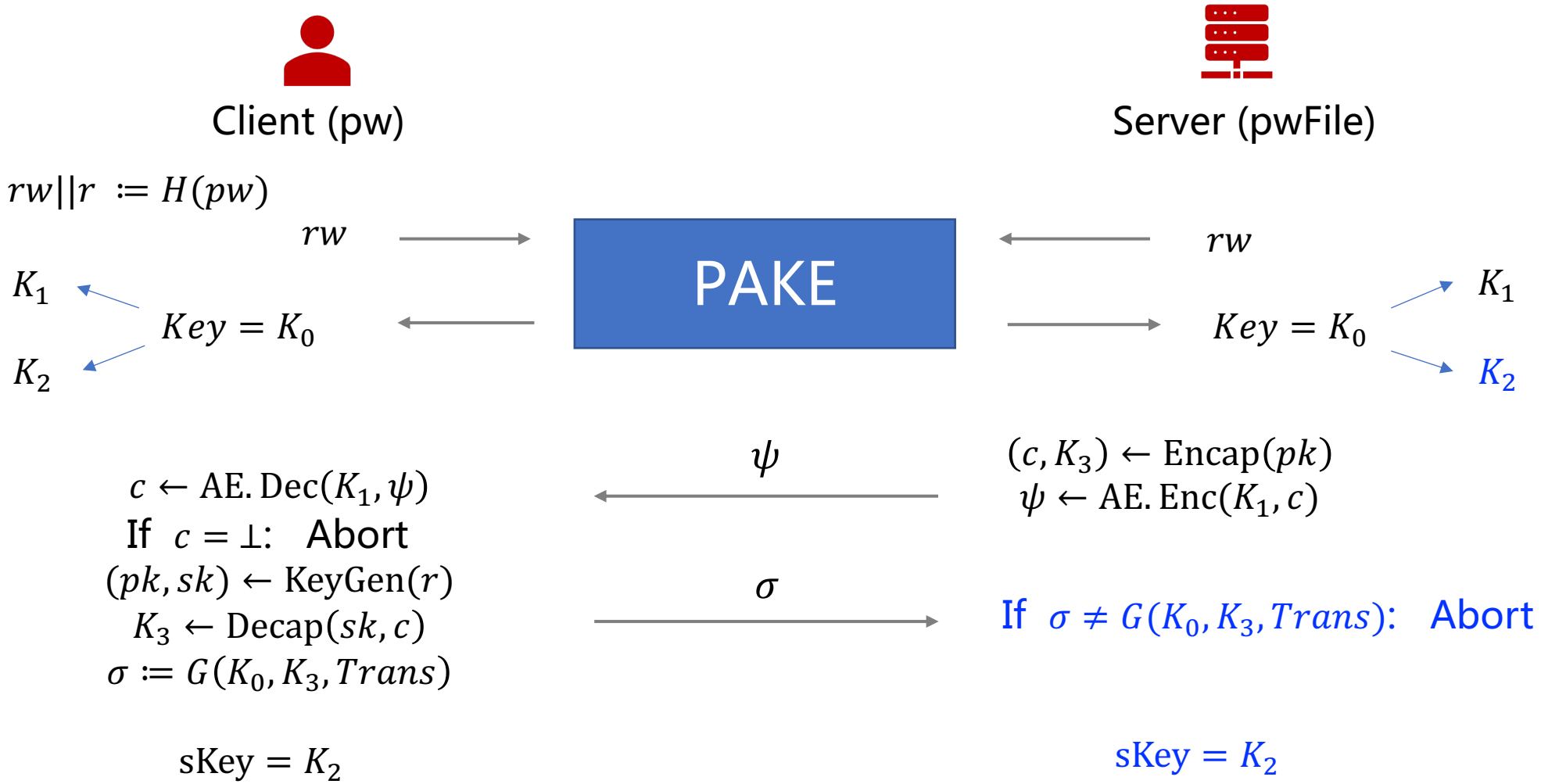
aPAKE compiler from KEM and AE

Registration: $pwFile = (rw, pk)$, where $rw||r = H(pw)$, $(pk, sk) \leftarrow \text{KEM.KeyGen}(r)$

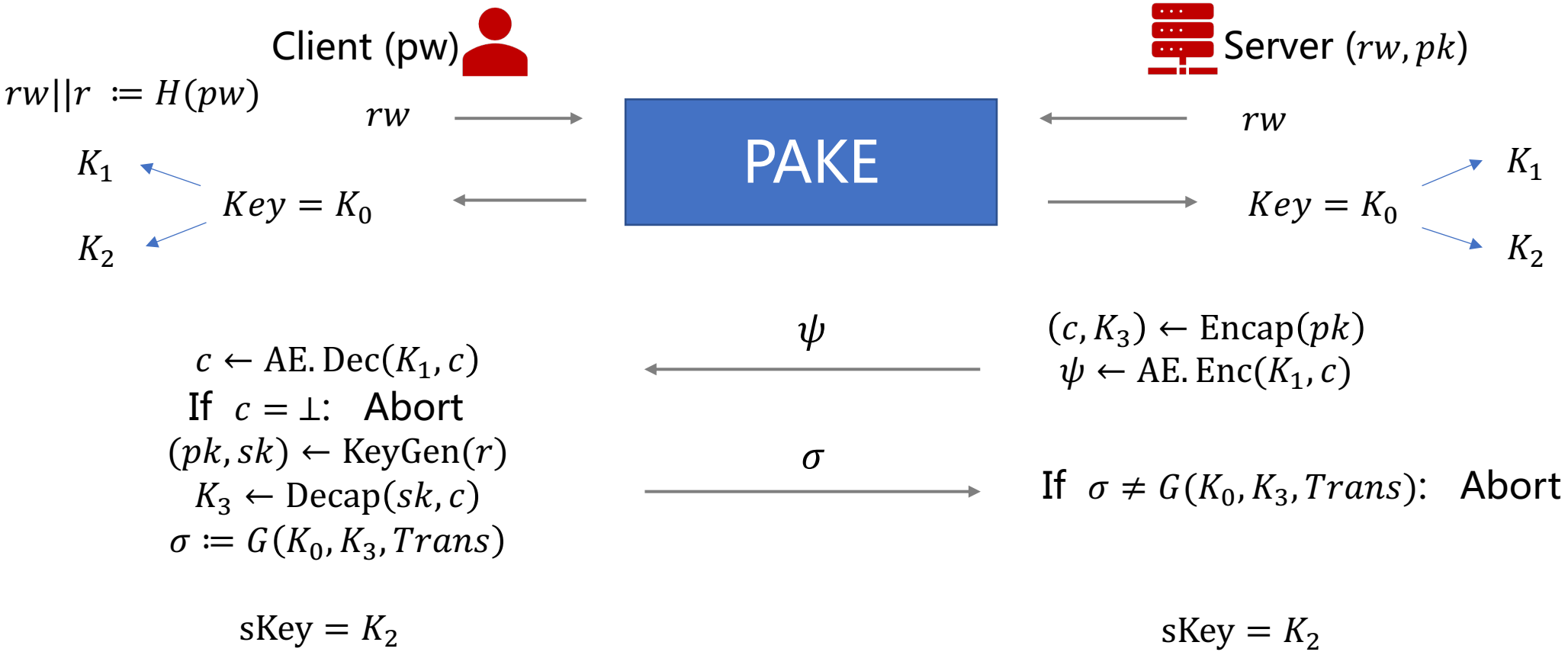


aPAKE compiler from KEM and AE

Registration: $pwFile = (rw, pk)$, where $rw||r = H(pw)$, $(pk, sk) \leftarrow \text{KEM.KeyGen}(r)$

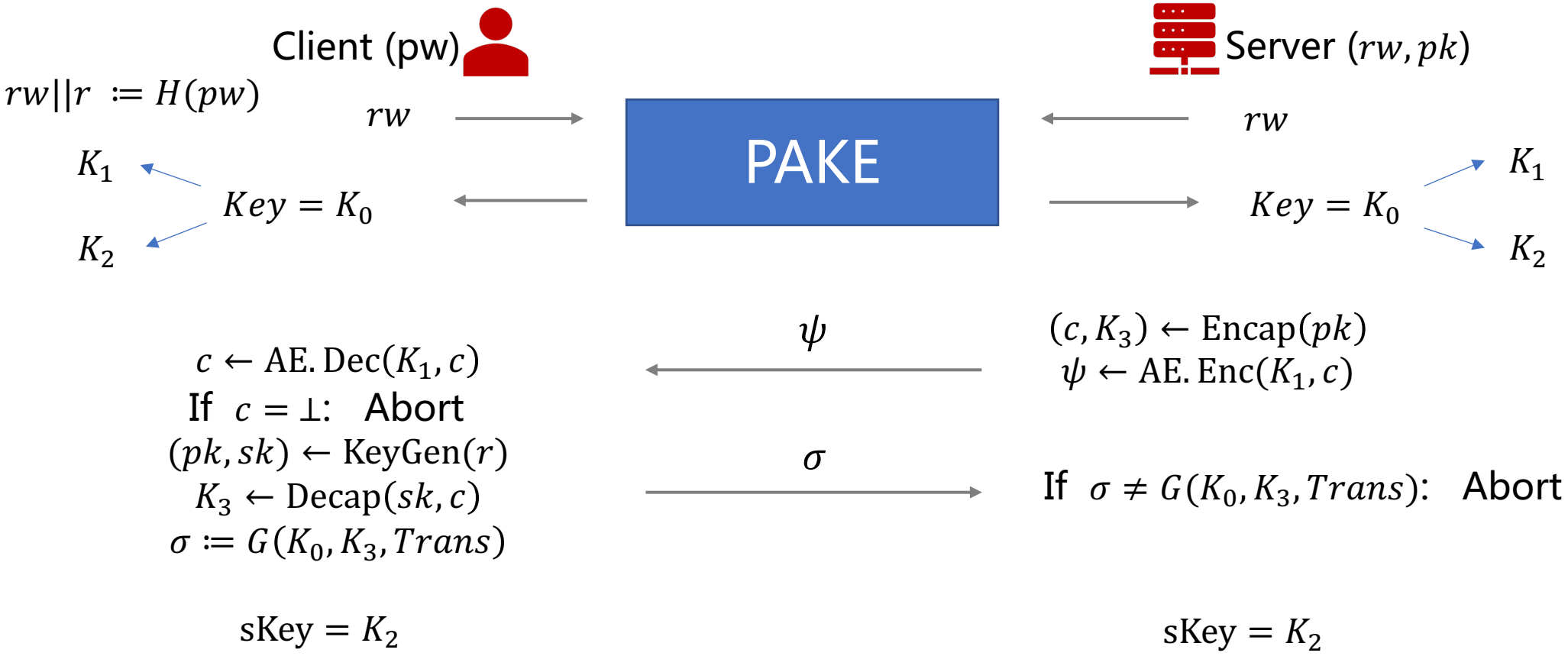


Security Analysis: Sim construction



- Sim can simulate **indistinguishable transcript** of aPAKE protocol in the real world.
- Sim can simulate **indistinguishable output session key** for each client/server.

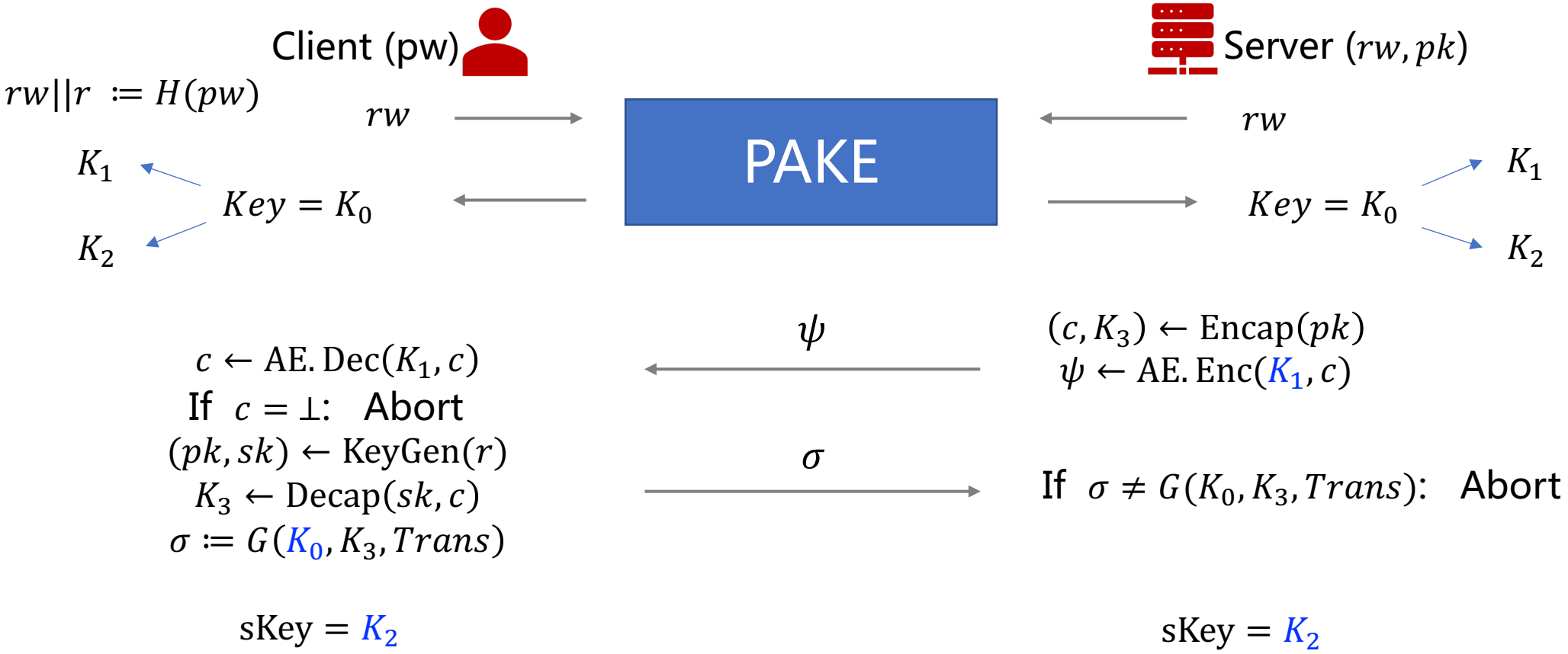
Security Analysis: before server is compromised



Case I: $\mathcal{A}$ guesses the password rw $\Rightarrow$ Sim can extract the correct pw from rw with the help of RO

With the correct pw, Sim can simulate the transcript and session key perfectly

Security Analysis: before server is compromised

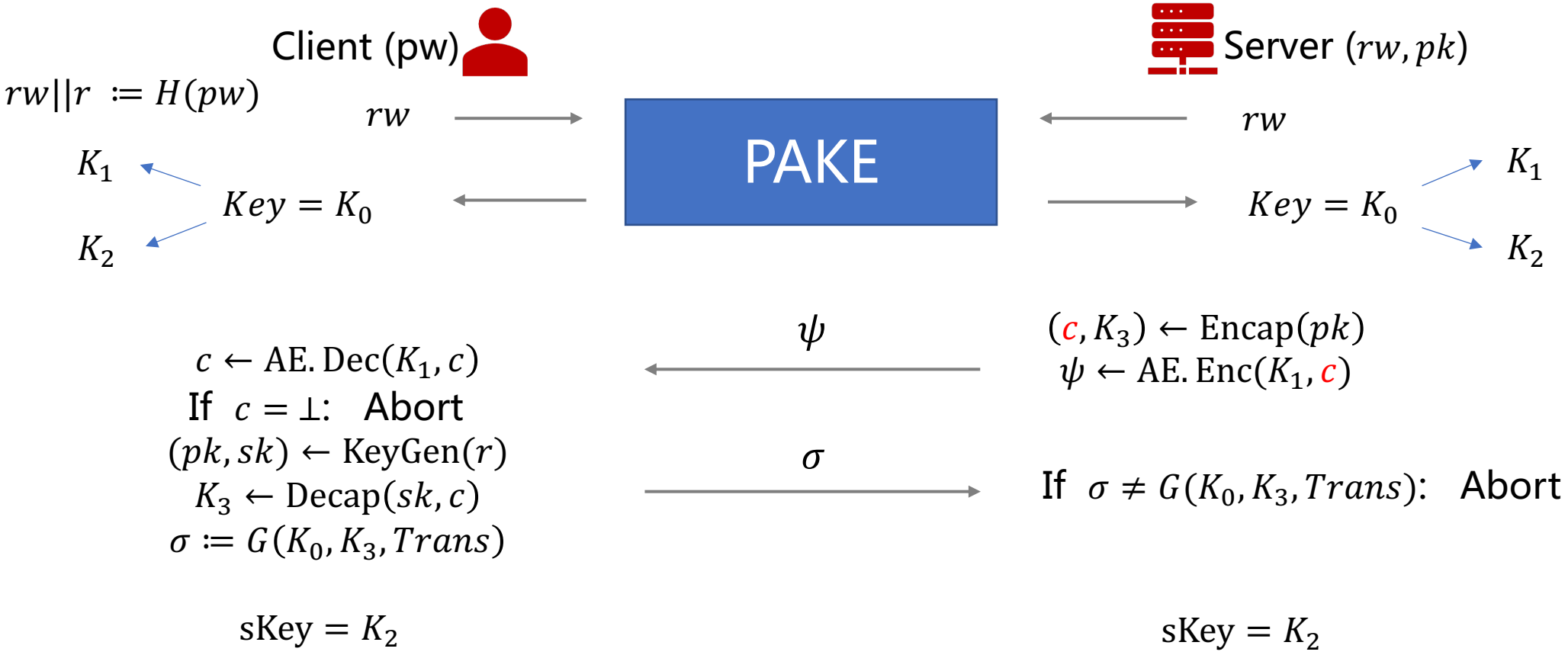


Case II: $\mathcal{A}$ guesses a wrong password $rw \neq H_0(pw)$ $\Rightarrow$ K_0, K_1, K_2 become uniform

Then the message ψ and σ leak no information of $pw, rw,$ or pk

The security of session key is guaranteed by the uniformity of K_2

Security Analysis: before server is compromised

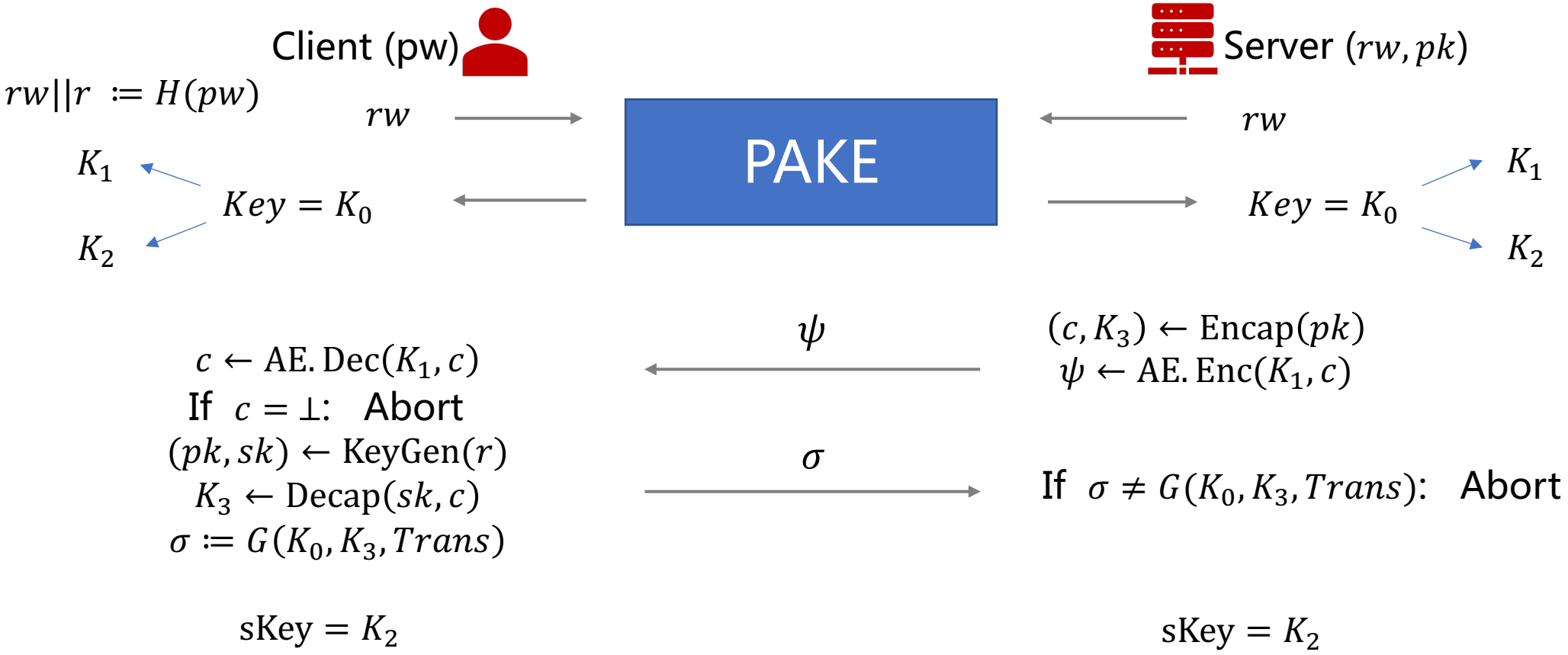


Case II: $\mathcal{A}$ guesses a wrong password $rw \neq H_0(pw)$ $\Rightarrow$ K_0, K_1, K_2 become uniform

The necessity of AE: protect the password information leakage in ciphertext c

For each pw, $\mathcal{A}$ computes $(pk, sk) \leftarrow \text{KeyGen}(r)$, and checks whether c is a valid encapsulation under pk

Security Analysis: after server is compromised

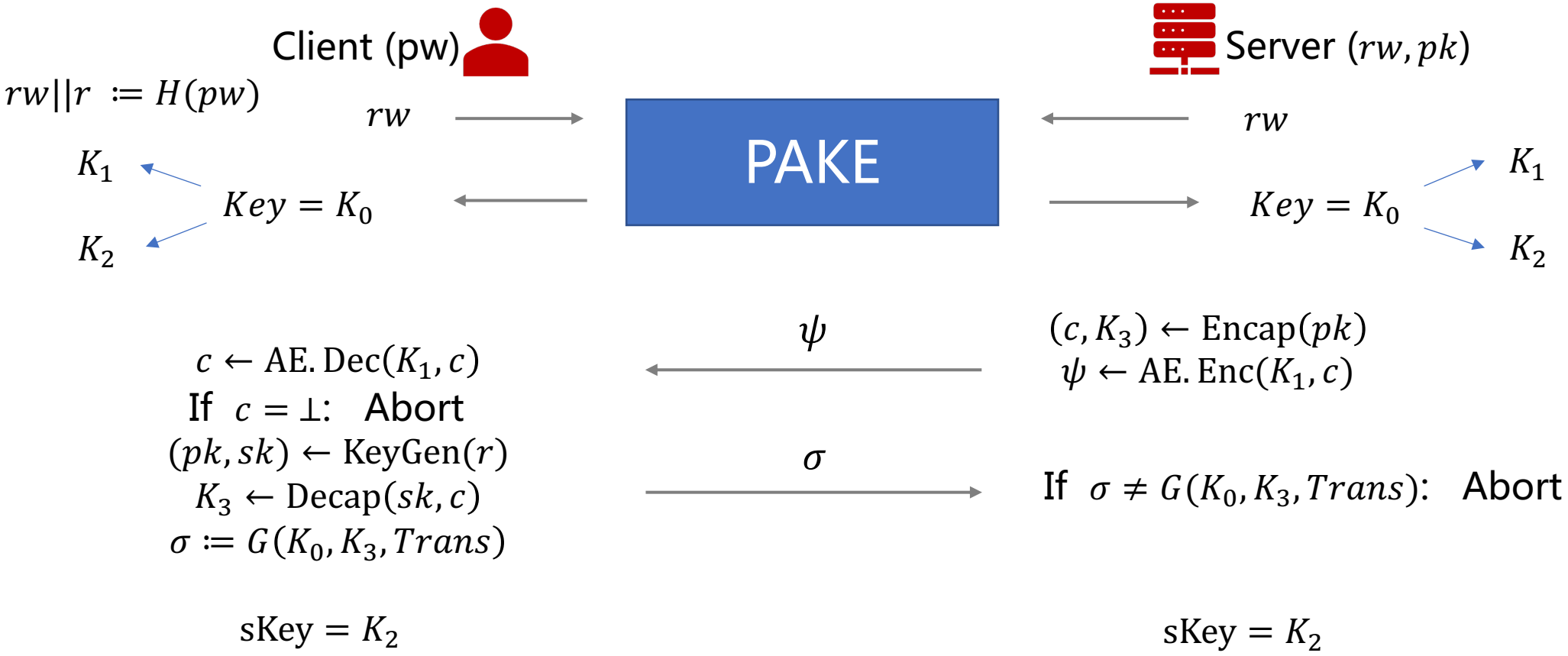


$\mathcal{A}$ cannot get information of pw from previous session due to the forward security of PAKE

$\mathcal{A}$ gets the password file, and hence it can impersonate the server

We need to show: $\mathcal{A}$ cannot impersonate the client unless it queries $H(pw)$ for the correct pw

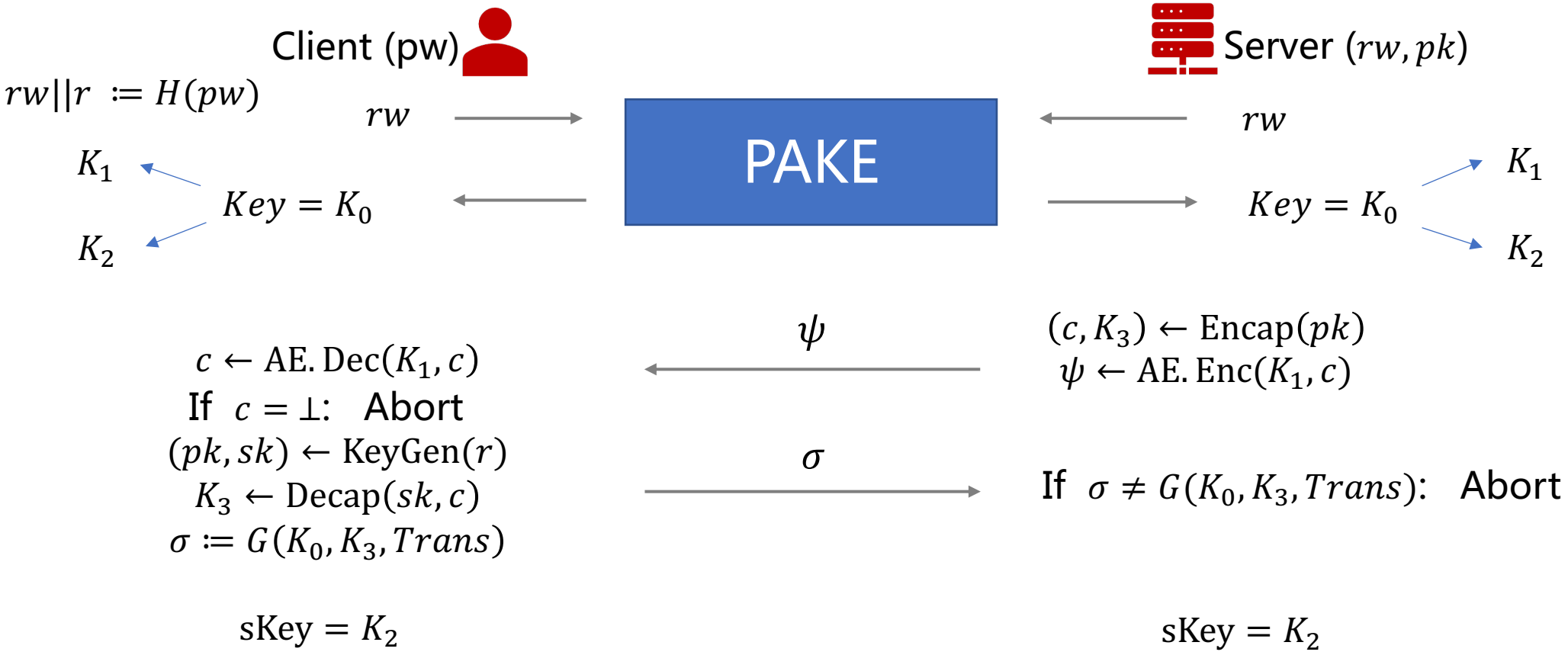
Security Analysis: after server is compromised



We need to show: $\mathcal{A}$ cannot impersonate the client unless it queries $H(pw)$ for the correct pw

High level idea: if $\mathcal{A}$ does not query $H(pw)$ but sends a valid $\sigma = G(K_0, K_3, Trans)$, it breaks the OW-PCA security of the underlying KEM

Security Analysis: aPAKE guarentee

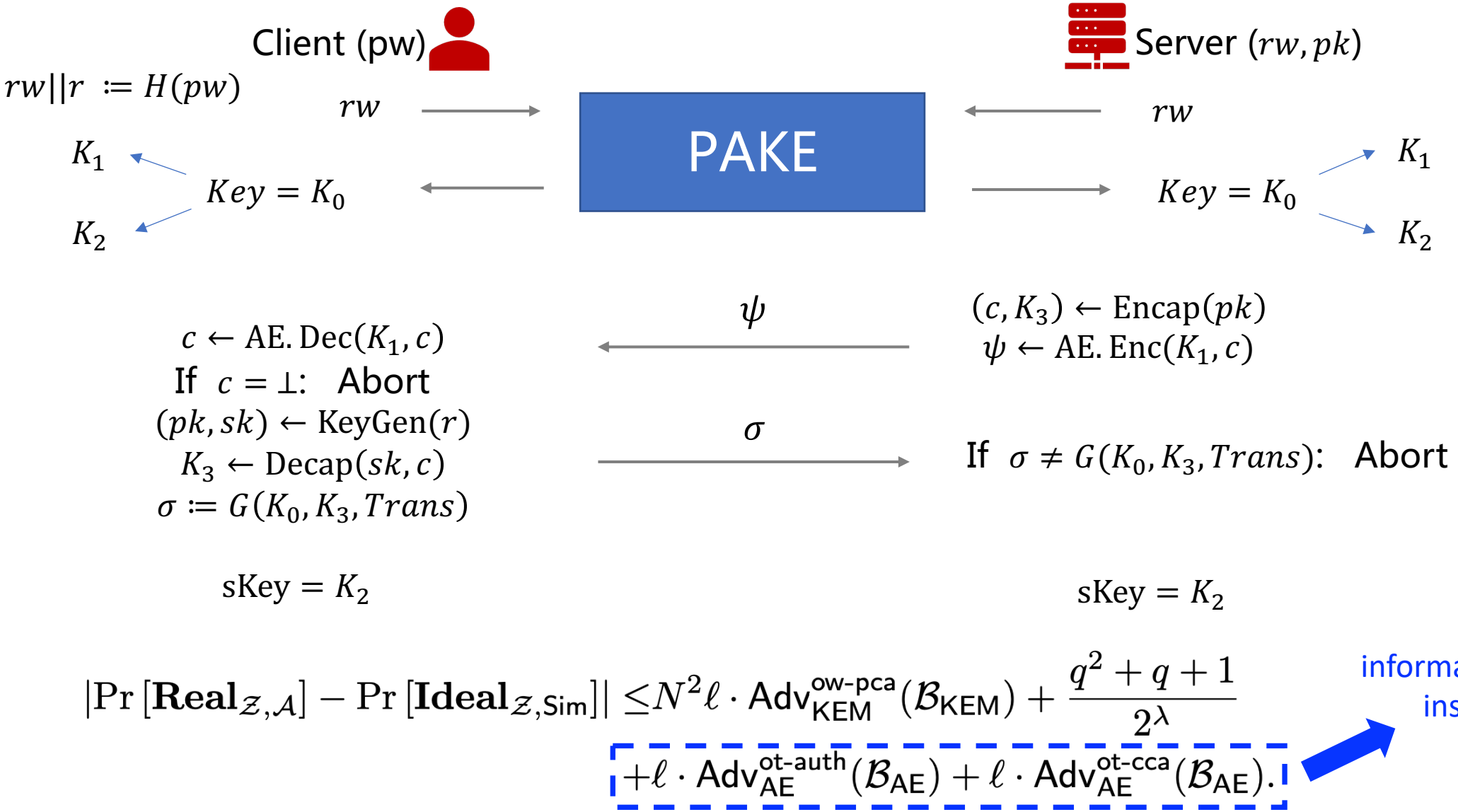


$\mathcal{A}$ cannot impersonate the client unless it queries $H(pw)$ for the correct pw

Sim can simulate the transcript and session key without knowledge of pw

$\mathcal{A}$ can only try different pw and compute $H(pw)$, using a brute-force search

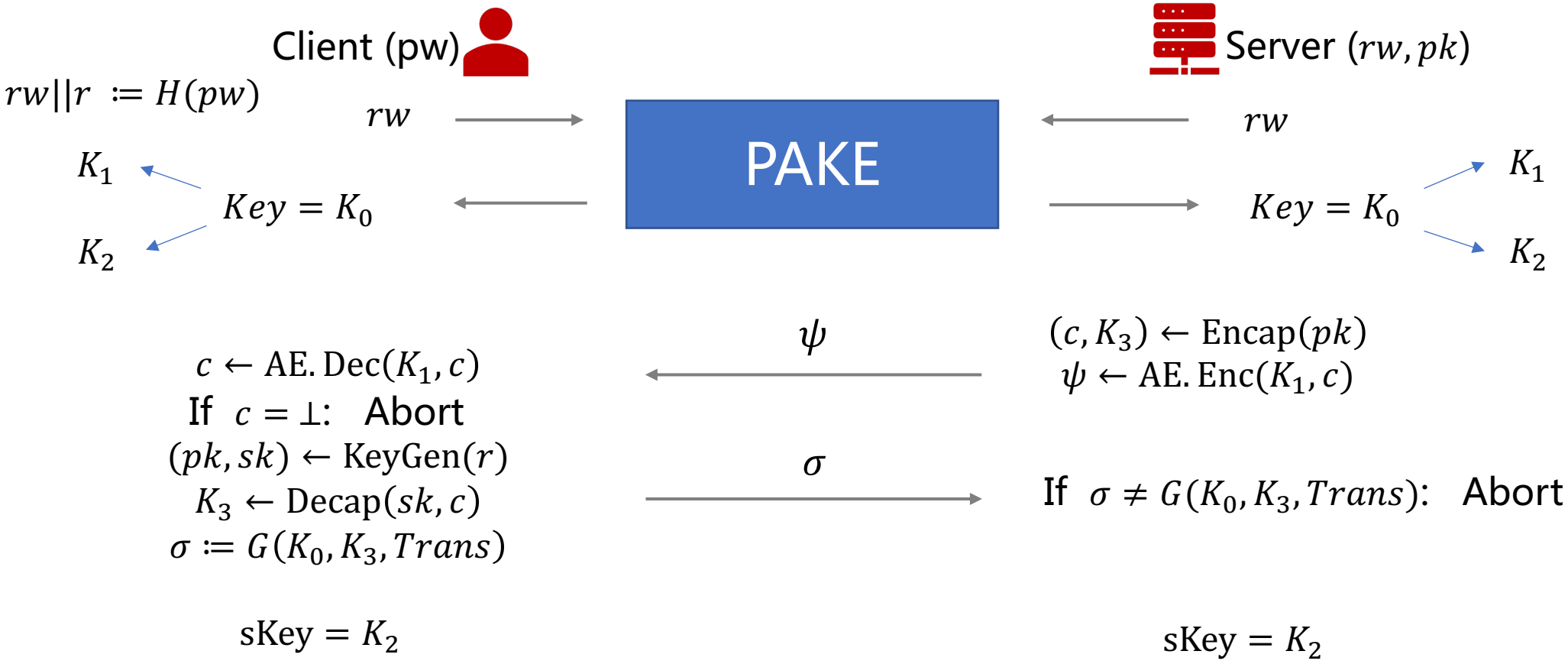
Security Analysis: Tightness



$$|\Pr[\mathbf{Real}_{\mathcal{Z},\mathcal{A}}] - \Pr[\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}}]| \leq N^2\ell \cdot \mathsf{Adv}^{\text{ow-pca}}_{\mathsf{KEM}}(\mathcal{B}_{\mathsf{KEM}}) + \frac{q^2+q+1}{2^\lambda}$$
$$+\ell \cdot \mathsf{Adv}^{\text{ot-auth}}_{\mathsf{AE}}(\mathcal{B}_{\mathsf{AE}}) + \ell \cdot \mathsf{Adv}^{\text{ot-cca}}_{\mathsf{AE}}(\mathcal{B}_{\mathsf{AE}}).$$

information-theoretical instantiations

Security Analysis: Tightness



$$|\Pr[\mathbf{Real}_{\mathcal{Z},\mathcal{A}}] - \Pr[\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}}]| \leq \mathsf{Adv}_{\mathsf{KEM}}^{(N^2,\ell)\text{-ChCCA}}(\mathcal{B}_{\mathsf{KEM}}) + \frac{2\ell + q^2 + q + 1}{2^\lambda}.$$

Contents

Efficiency Comparison

Protocol	Computing Time (ms)	Communication Cost (KB)	Assumption
[GMR06] + Dilithuim	0.337 + 0.214 = 0.551	3.871 + 4.657 = 8.528	Lattice
[GMR06] + Falcon	0.363 + 0.162 = 0.525	1.298+2.001 = 3.299	Lattice
OPAQUE	0.280 + 0.230 = 0.51	0.156+0.312 = 0.468	OM-DH
Ours + Kyber	0.143 + 0.171 = 0.314	0.687+1.813 = 2.5	Lattice

We use lattice-based EKE-PAKE as our underlying PAKE instantiation.

- ✓ Ours is the most efficient one.
- ✓ Communication Cost of ours is the second best.

The experiments are obtained by running experiments on Intel(R) Core(TM) i7-1068NG7 CPU @ 2.30GHz with 4 cores under macOS 13.3.1.

Conclusion & future direction

Conclusion

- We design an efficient aPAKE compiler instantiable from lattices
- Our compiler has UC security, good round efficiency and provides mutual explicit authentication

Future direction

- Lattice-based strong aPAKE protocols ?
- aPAKE protocols in QROM? (impossible results, or new security model in QROM)

For more information, please refer to the full version our paper.

https://eprint.iacr.org/2024/1400.pdf

Thanks! Questions?