KHAPE: Asymmetric PAKE from Key-Hiding Authenticated Key Exchange

Crypto 2021

Yanqi Gu

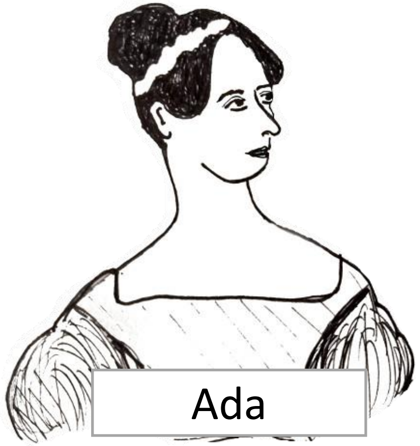
Stanislaw Jarecki

Hugo Krawczyk

PAKE: Password Authenticated Key Exchange

[BM92,…,BPR00,BMP00,…,GL01,KOY01,…,CHKLM05,…]

Authentication in the symmetric password-only setting



$pw \longrightarrow$ **PAKE** $\longleftarrow pw'$

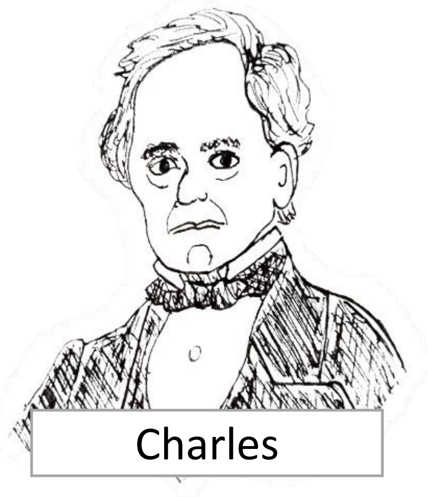
$k \longleftarrow$ $\qquad$ $\longrightarrow k'$

PAKE

$k = k'(\leftarrow \$)$

if $pw = pw'$

else k, k' independent



Most efficient PAKE's: "password-blinded" Diffie-Hellman Key Exchange (DH KE):

EKE [BM92], *SPEKE* [Jab96], *PPK* [BMP00], *SPAKE2* [AP05], *TBPEKE* [PW17], *CPace* [HL19], …

Low overhead over DH KE $\rightarrow$ cost(PAKE) $\approx$ cost(KE)

(for cost-comparisons we assume prime-order groups + ROM)

asymmetric PAKE (aPAKE): for client-server setting

[Jablon97,…,GMR06,…,BP13,…]

Server Initialization: on input pw, compute a "hashed password", $hpw = \mathrm{H}(pw)$

H : one-way hash

$pw \rightarrow$

$k \leftarrow$

aPAKE

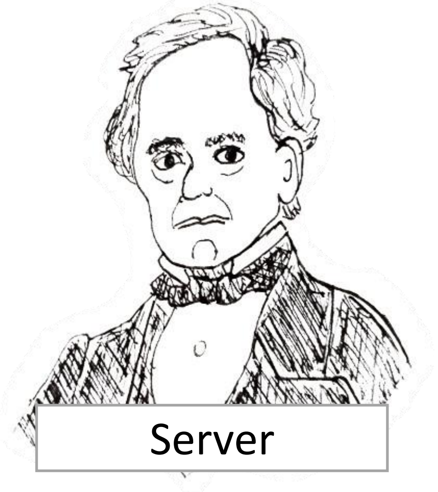
$k = k'(\leftarrow \$)$

if $hpw = \mathrm{H}(pw)$

else k, k' independent

$\leftarrow hpw$

$k' \rightarrow$



Benefit of aPAKE: server compromise leaks $pw = \mathrm{H}^{-1}(hpw)$ only via brute-force search

Most efficient aPAKE's: PAKE + pw-based PKE [BMP00,…,HJKLSX18,Shoup20] or Signature […,GMR06,HJKLSX18]

cost(aPAKE) $\approx$ 1.5 x cost(KE), using Schnorr signatures

(for cost-comparisons we assume prime-order groups + ROM)

strong asymmetric PAKE (saPAKE)

[JKX'18,BJX'19,JKX'21]

Server Initialization: on input pw, pick random "salt" s, compute $hpw = H_s(pw)$

H : keyed one-way hash

$pw \rightarrow$

aPAKE

$k = k'(\leftarrow \$)$

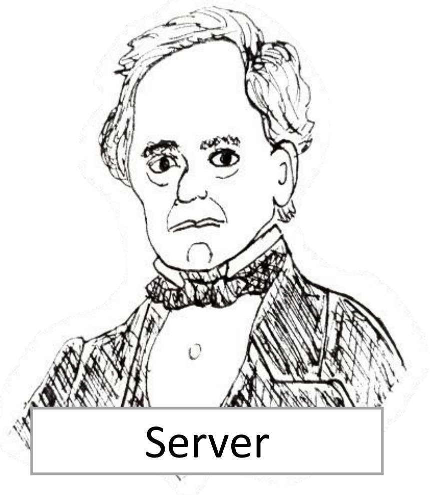
if $hpw = H_s(pw)$

else k, k' independent

$\leftarrow s, hpw$

$k \leftarrow$

$k' \rightarrow$



Benefit of saPAKE: the brute-force attack on server compromise cannot be precomputed

Most efficient saPAKE: *OPAQUE* = (OPRF + AKE) [JKX18]

cost(saPAKE) = cost(OPRF) + cost(AKE) $\approx$ 2 x cost(KE) [JKX18,JKX21]

(for cost-comparisons we assume prime-order groups + ROM)

Authenticated Key Exchange (AKE) [Br93,Kra96,CK01,Kra03,…]

public key setting: each party generates (sk,pk)

Public keys pkC and pkS assumed known by both Client and Server

$skC \rightarrow$ AKE $\leftarrow skS$

AKE

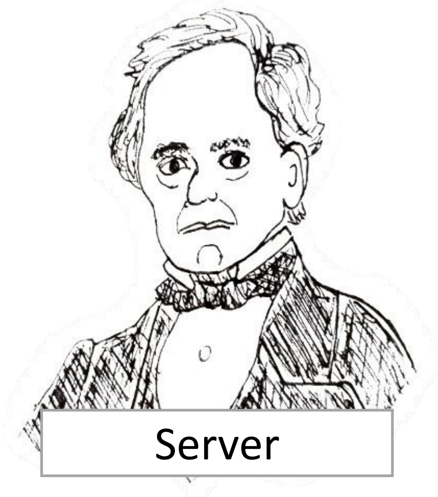
$$k = k' \ (\leftarrow \$)$$

If $\mathrm{pk}(skC) = pkC$

and $\mathrm{pk}(skS) = pkS$

else k, k' independent

$\leftarrow k$ (to Client) $\qquad k' \rightarrow$ (to Server)



Most efficient AKE: *HMQV* [Kra05]

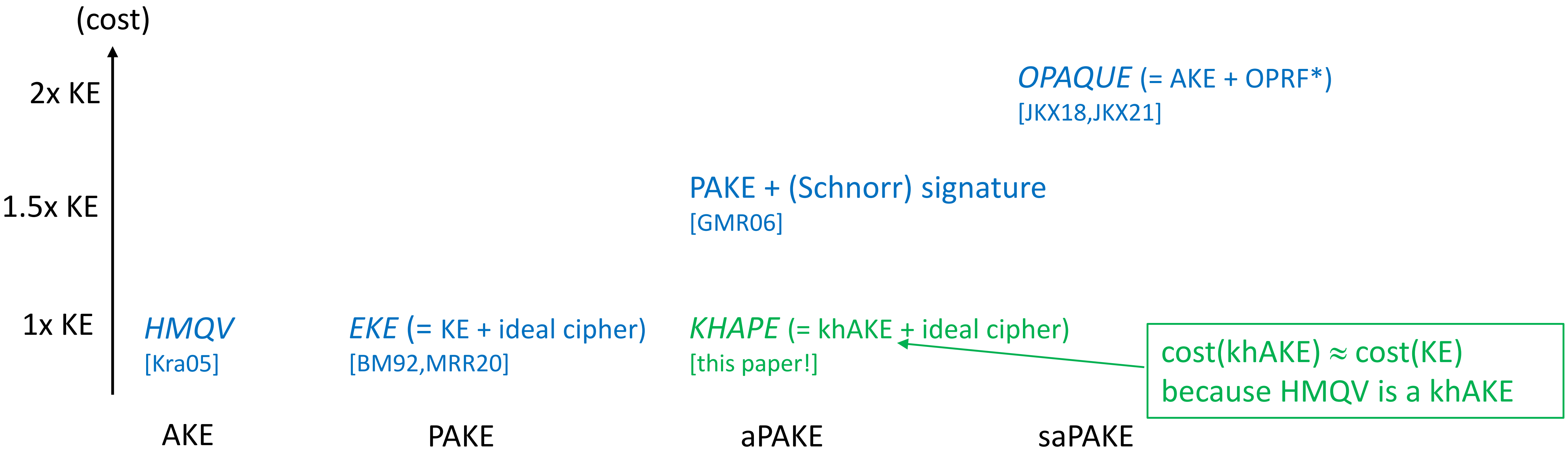
cost(AKE) $\approx$ cost(KE)

(for cost-comparisons we assume prime-order groups + ROM)

must aPAKE cost more than KE / AKE / PAKE?

comparisons for prime-order groups, assuming DH (OneMore-DH for *) and ROM

our main contribution: *KHAPE*: aPAKE from *key-hiding* AKE (khAKE)



AKE:	Authenticated KE	*parties hold public keys*
PAKE:	Password-Authenticated KE	*parties hold shared password*
aPAKE:	asymmetric PAKE	*server holds password hash*
saPAKE:	strong asymmetric PAKE	*server holds randomized password hash*

our contributions

1. *KHAPE*: generic construction of aPAKE = IdealCipher + key-hiding AKE
   - assumes Ideal Cipher on (elliptic curve) group
     - option 1: uniform EC point encoding, e.g. Elligator2 [BHKT13] or Elligator^2 [Tib14]
     - option 2: “weak IC” of [MRR20], one RO-indifferentiable hash onto curve per enc/dec *[still to confirm]*

2. UC notion of key-hiding AKE (khAKE)
   - realized (in ROM) by *HMQV* [Kra03], *3DH* [MP16], *SKEME* [Kra96] (first proofs of UC security for these AKEs)
   - *SKEME* is a generic construction → post-quantum aPAKE from lattice KEM (alternative to e.g. [GMR06,HJKLSX18])

→ cost(aPAKE) ≈ cost(KE) (if KHAPE is instantiated with khAKE = *HMQV)*

- price: 4 flows (3 if S starts) vs 3 flows of *SPAKE2+* [APE05,Shoup20], *GMR+EKE* [GMR06], *OPAQUE* [JKX18]

→ efficient instantiation of saPAKE construction *OPAQUE’* (= OPRF + aPAKE) [JKX18]

- alternative to *OPAQUE* (= OPRF + AKE) [JKX18]
- advantages: graceful security degradation from saPAKE to aPAKE if OPRF key leaks
  application examples: outsourced/distributed OPRF, quantum-insecure OPRF
- disadvantages: added protocol flow, need *key-hiding* AKE (e.g. cannot use *Sigma* AKE [Kra03] from TLS)

UC notion of key-hiding AKE (khAKE)

public key setting: each party generates (sk,pk)

your own secret key

counterparty's presumed public key

skC , pkS →

Client

$\leftarrow k$

khAKE

$$k = k' \ (\leftarrow \$)$$

If $\quad \mathrm{pk}(skC) = pkC$

and $\mathrm{pk}(skS) = pkS$

else k, k' independent

← pkC , skS

$k' \rightarrow$

Server

Key-Hiding AKE properties (assume attacker plays client's role):

- Server's output k' reveals only if adversary's values (skC^*, pkS^*) matches Server's inputs (pkC, skS)
- **no** *perfect-forward secrecy (pfs) / privacy (pfp)*: k' can be computed for (skC^*, pkS^*) leaked <u>in the future</u>
    - pfs: easy to achieve with key confirmation messages
    - pfp: harder (can be achieved with less efficient protocols, e.g. "input-commit + SPHF")

[why not include pfs/pfp? we model minimal khAKE properties sufficient for our khAKE-to-aPAKE compiler]

key-hiding AKE example 1: *3DH* [MP16]

Diffie-Hellman KE/KEM ≈ ElGamal PKE

Client Init: $skC \leftarrow Z_p$, $pkC = g^{skC}$

Server Init: $skS \leftarrow Z_p$, $pkS = g^{skS}$

Input: skC, pkS

Client

Input: pkC, skS

Server

$x \leftarrow Z_p$

$X = g^x$ $\longrightarrow$

$\longleftarrow$ $Y = g^y$

$y \leftarrow Z_p$

Output computation: $k = \mathrm{H}[\ \mathrm{DH}(X, pkS)\ , \mathrm{DH}(pkC, Y)\ , \mathrm{DH}(X, Y)\ ]$

- H is an RO hash (also taking session and party identifiers as inputs)
- DH is a Diffie-Hellman KE function, i.e. $DH\left(g^a, g^b\right) = g^{a \cdot b}$

key-hiding AKE example 2: *HMQV* [Kra05]

Diffie-Hellman KE/KEM ≈ ElGamal PKE

Client Init: $skC \leftarrow Z_p$, $pkC = g^{skC}$

Server Init: $skS \leftarrow Z_p$, $pkS = g^{skS}$

Input: skC, pkS

Client

$x \leftarrow Z_p$

$X = g^x$ →

Input: pkC, skS

Server

$y \leftarrow Z_p$

← $Y = g^y$

Output computation: $k = \mathrm{H}\big[\ \mathrm{DH}\big(X \cdot (pkC)^{H'(X)}\ ,\ Y \cdot (pkS)^{H'(Y)}\big)\ \big]$

- H is an RO hash (also taking session and party identifiers as inputs)
- DH is a Diffie-Hellman KE function, i.e. $DH\left(g^a, g^b\right) = g^{a \cdot b}$

cost(HMQV)
≈ cost(DH KE)

KHAPE: aPAKE from key-hiding AKE + Ideal Cipher

Initialization: $(skC, pkC) \leftarrow AKE.KG,\ (skS, pkS) \leftarrow AKE.KG$
$env \leftarrow IC.Enc_{pw}(skC, pkS)$ group element

Input: pw

Client

Input: $hpw = (env, pkC, skS)$

Server

$\xleftarrow{\ env\ }$

$(skC, pkS) \leftarrow IC.Dec_{pw}(env)$

$(skC, pkS) \rightarrow$ **khAKE** $\leftarrow (pkC, skS)$

$\xleftarrow{\ k\ }$ khAKE $\xrightarrow{\ k'\ }$

key confirmation messages

$\tau = PRF_k(1)$ $\longrightarrow$

$\gamma = PRF_{k'}(2)$ $\longleftarrow$

Compare to "EKE" PAKE of [BM92]:

- EKE = KE $\rightarrow$ PAKE compiler
- KHAPE = AKE $\rightarrow$ aPAKE compiler

- EKE uses IC to pw-encrypt <u>each KE message</u>
- KHAPE uses IC to pw-encrypt <u>C's AKE inputs</u>

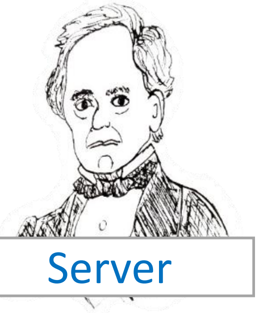
KHAPE: aPAKE from key-hiding AKE + Ideal Cipher

Initialization: $(skC, pkC) \leftarrow AKE.KG,\ (skS, pkS) \leftarrow AKE.KG$
$env \leftarrow IC.Enc_{pw}(\, skC, pkS\,)$

Input: pw

Input: $hpw = (env, pkC, skS)$

Client



Server

$\xleftarrow{\ env\ }$

$(skC, pkS) \leftarrow IC.Dec_{pw}(env)$

$(skC, pkS) \longrightarrow$ khAKE $\longleftarrow (pkC, skS)$

$k \longleftarrow$ khAKE $\longrightarrow k'$

$\xrightarrow{\ \tau = PRF_k(1)\ }$

$\xleftarrow{\ \gamma = PRF_{k'}(2)\ }$

For passive attacker:

- IC $\rightarrow$ each pw^* maps to \$ (skC^*, pkS^*)

If attacker plays server role:

- IC $\rightarrow$ ciphertext env^* commits to single pw^* s.t. $IC.Dec_{pw*}(env^*)$ is non-random

$\rightarrow$ single pkS^* for which Adv knows skS^*

$\rightarrow$ single pw^* on which Adv attacks the Client

KHAPE: aPAKE from key-hiding AKE + Ideal Cipher

Initialization: $(skC, pkC) \leftarrow AKE.KG$, $(skS, pkS) \leftarrow AKE.KG$
$env \leftarrow IC.Enc_{pw}(\, skC, pkS\,)$

Input: pw

Client

Input: $hpw = (env, pkC, skS)$

Server

env ←

$(skC, pkS) \leftarrow IC.Dec_{pw}(env)$

(skC, pkS) → khAKE ← (pkC, skS)

k ← khAKE → k'

$\tau = PRF_k(1)$ →

$\gamma = PRF_{k'}(2)$ ←

For passive attacker:

- IC $\rightarrow$ each pw^* maps to \$ (skC^*, pkS^*)

If attacker plays the server role:

- IC $\rightarrow$ ciphertext env^* commits to single pw^* s.t. $IC.Dec_{pw*}(env^*)$ is non-random

$\rightarrow$ single pkS^* for which Adv knows skS^*

$\rightarrow$ single pw^* on which Adv attacks the Client

Can we omit Client's key conf msg τ ?

If attacker plays the client role:

- khAKE block has **no Perfect Forward Secrecy**

$\rightarrow$ Adv could then test (skC^*, pkS^*) against S's key

$\rightarrow$ Adv could then test off-line test any pw^*

- C's key conf msg commits to single (skC^*, pkS^*)

$\rightarrow$ C's key conf msg commits to single pw^*

KHAPE: aPAKE from key-hiding AKE + Ideal Cipher

Initialization: $(skC, pkC) \leftarrow AKE.KG,\ (skS, pkS) \leftarrow AKE.KG$
$env \leftarrow IC.Enc_{pw}(skC, pkS)$

Input: pw

Client

Input: $hpw = (env, pkC, skS)$

Server

$\xleftarrow{env}$

$(skC, pkS) \leftarrow IC.Dec_{pw}(env)$

$(skC, pkS) \rightarrow$ **khAKE** $\leftarrow (pkC, skS)$

$k \leftarrow$ khAKE $\rightarrow k'$

$\xrightarrow{\tau = PRF_k(1)}$

$\xleftarrow{\gamma = PRF_{k'}(2)}$

For passive attacker:

- IC $\rightarrow$ each pw^* maps to \$ (skC^*, pkS^*)

If attacker plays server role:

- IC $\rightarrow$ ciphertext env^* commits to single pw^* s.t. $IC.Dec_{pw^*}(env^*)$ is non-random

$\rightarrow$ single pkS^* for which Adv knows skS^*

$\rightarrow$ single pw^* on which Adv attacks the Client

Can we omit Server's key conf msg γ?

If attacker plays the server role:

- khAKE block has **no Perfect Forward Secrecy**

$\rightarrow$ Adv can compute Client's key if it compromises Server and learns (pkC, skS) <u>in the future</u>

- S's key conf msg verifies Adv knows (pkC, skS) <u>before</u> Client outputs a key

conclusions & some follow-up questions

Our contributions:

- *KHAPE,* new aPAKE from key-hiding AKE
  - optimal computational efficiency ($\approx$ KE), black-box from KEM
- UC notion of key-hiding AKE, realized by *HMQV*, *3DH, SKEME*
- efficient instantiation of saPAKE protocol *OPAQUE'*
  - offers graceful security degradation from saPAKE to aPAKE if OPRF key leaks

Some follow-up questions:

- optimal-cost aPAKE with 3 flows? *[we have some results in preparation]*
- using lattice-based key-private KEMs for post-quantum security
  - Ideal Cipher for post-quantum?
  - Ideal Cipher for long bitstrings? *[we have some results in preparation]*
  - efficient post-quantum (UC) OPRF?

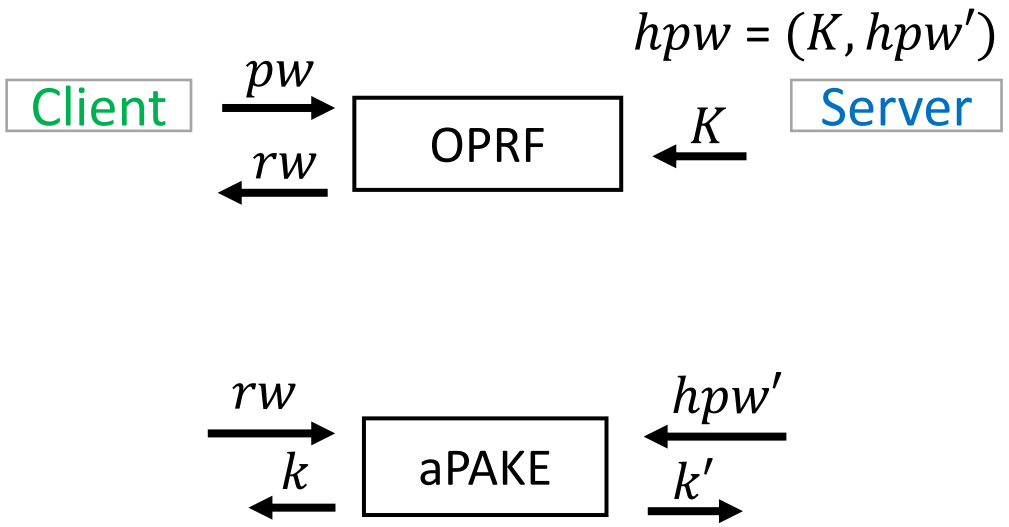
EXTRAS…

→ efficient instantiation of saPAKE *OPAQUE’* [JKX18]

$\boxed{\textit{OPAQUE'}\ \text{[JKX18]}}$ *(= OPRF + aPAKE)*

$K \leftarrow OPRF.KG$ (oblivious PRF)
$rw \leftarrow OPRF_K(pw)$
$hpw' \leftarrow aPAKE.Hash(rw)$

$hpw \leftarrow (k, hpw')$

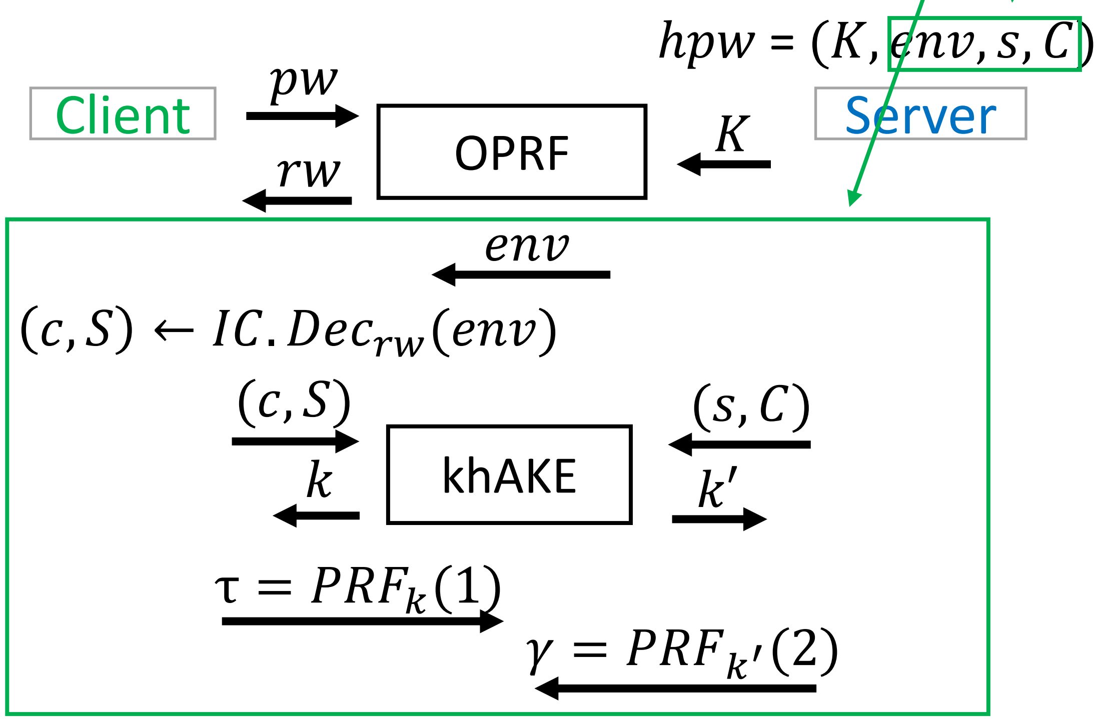


→ efficient instantiation of saPAKE *OPAQUE’* [JKX18]

OPAQUE’ [JKX18] *(= OPRF + aPAKE)*

KHAPE

$K \leftarrow OPRF.KG$ (oblivious PRF)
$rw \leftarrow OPRF_K(pw)$
$(c, C), (s, S) \leftarrow khAKE.KG$ (x2)
$env \leftarrow IC.Enc_{rw}(c, S)$
$hpw \leftarrow (K, env, s, C)$

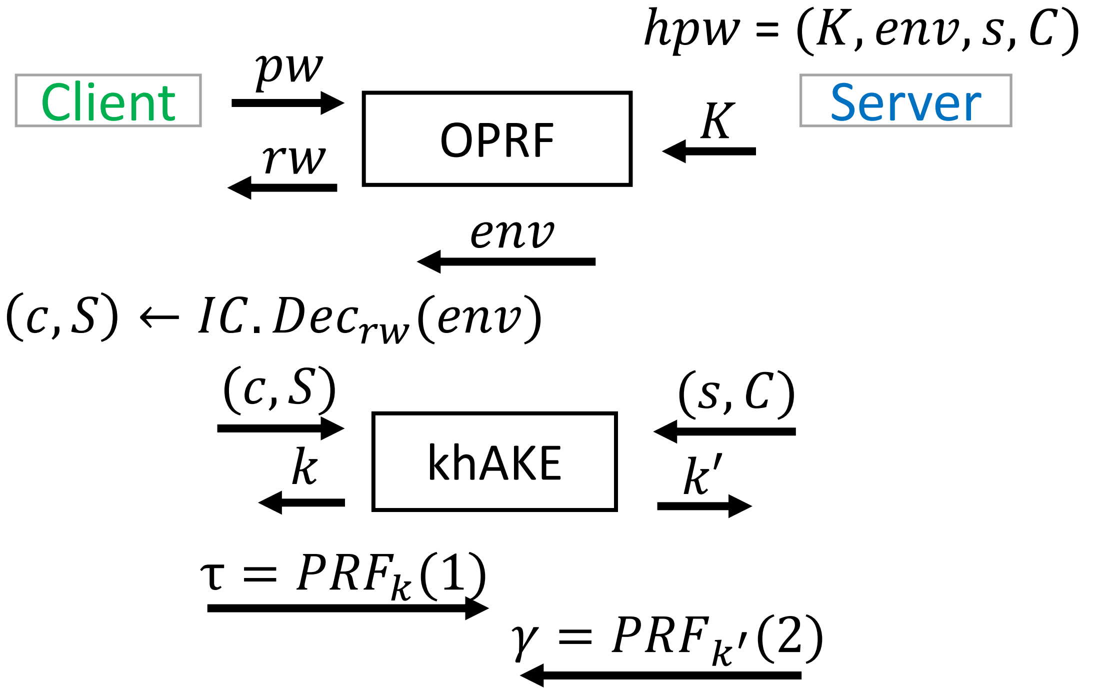


→ efficient instantiation of saPAKE *OPAQUE'* [JKX18]

OPAQUE' [JKX18] *(= OPRF + aPAKE)*

KHAPE

$K \leftarrow OPRF.KG$ (oblivious PRF)
$rw \leftarrow OPRF_K(pw)$
$(\mathrm{c}, \mathrm{C}), (\mathrm{s}, \mathrm{S}) \leftarrow khAKE.KG$ (x2)
$env \leftarrow IC.Enc_{rw}(c, S)$
$hpw \leftarrow (K, env, s, C)$



→ efficient instantiation of saPAKE *OPAQUE'* [JKX18]

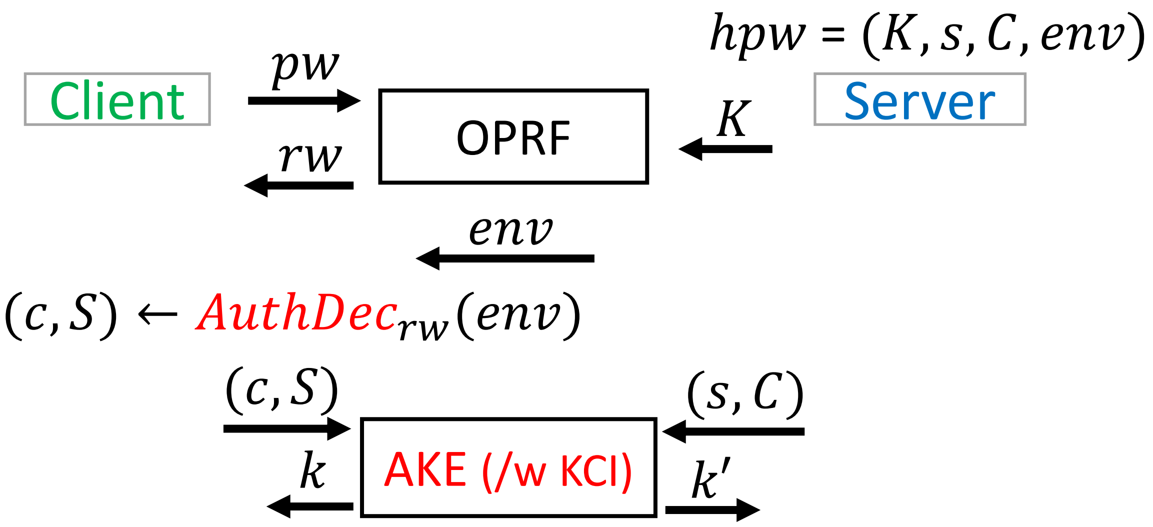
OPAQUE [JKX18] *(= OPRF + AuthEnc + AKE)*

$K \leftarrow OPRF.KG$
$rw \leftarrow OPRF_K(pw)$
$(\mathrm{c}, \mathrm{C}), (\mathrm{s}, \mathrm{S}) \leftarrow AKE.KG$ (x2)
$env \leftarrow AuthEnc_{rw}(c, S)$
$hpw \leftarrow (k, env, s, C)$

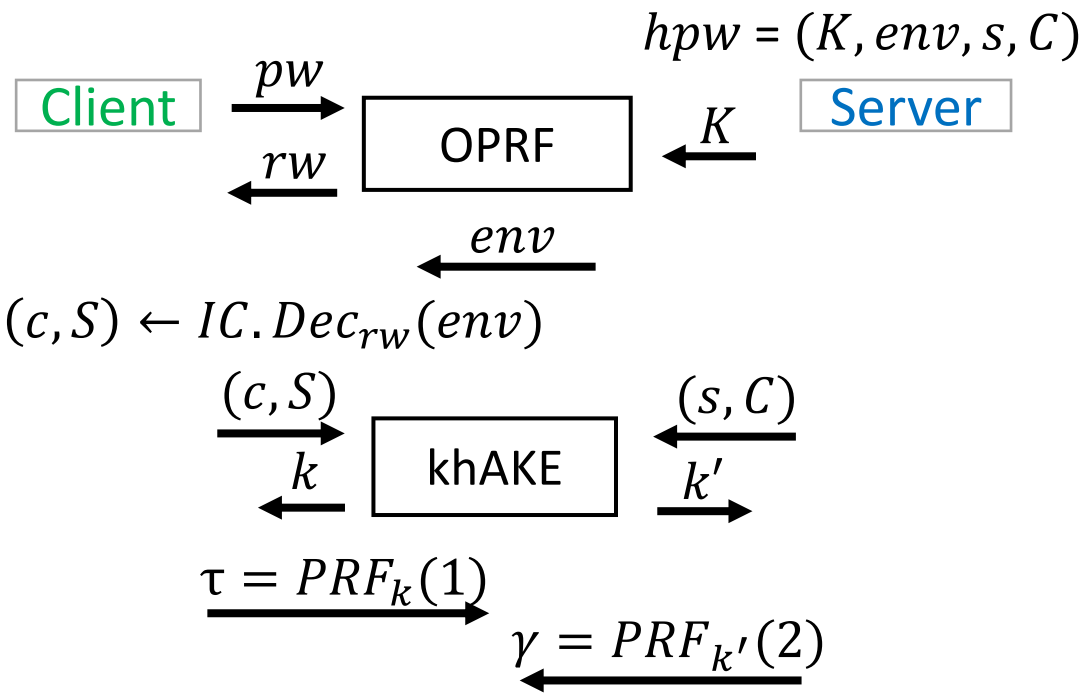
OPAQUE' [JKX18] *(= OPRF + aPAKE)*

KHAPE

$K \leftarrow OPRF.KG$ (oblivious PRF)
$rw \leftarrow OPRF_K(pw)$
$(\mathrm{c}, \mathrm{C}), (\mathrm{s}, \mathrm{S}) \leftarrow khAKE.KG$ (x2)
$env \leftarrow IC.Enc_{rw}(c, S)$
$hpw \leftarrow (K, env, s, C)$



**KCI = key compromise impersonation [security]*
≈ perf. forward secrecy ≈ key conf. flows



→ efficient instantiation of saPAKE *OPAQUE'* [JKX18]

OPAQUE [JKX18] *(= OPRF + AuthEnc + AKE)*

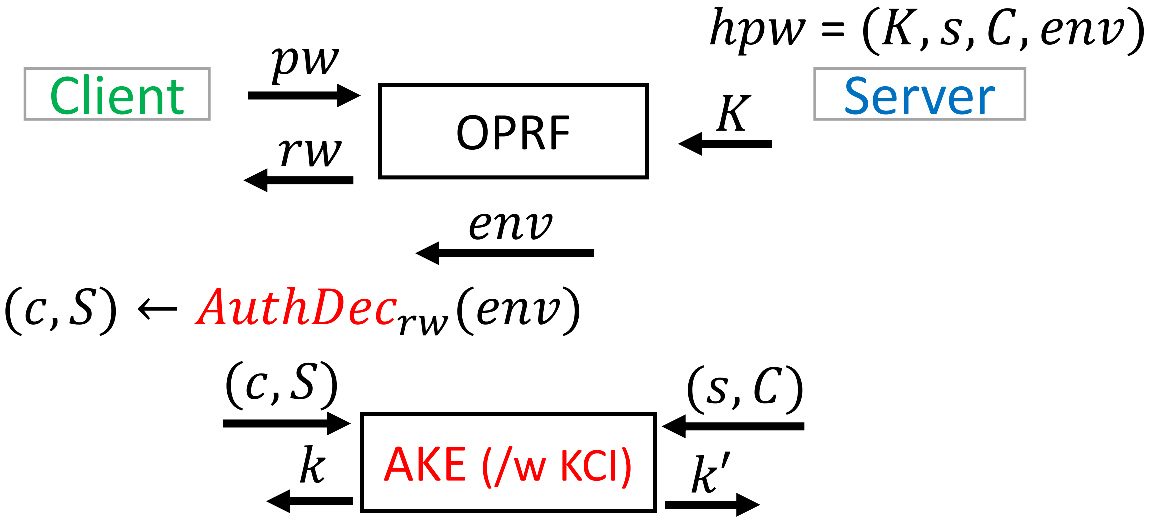
$K \leftarrow OPRF.KG$
$rw \leftarrow OPRF_K(pw)$
$(c, C), (s, S) \leftarrow AKE.KG$ (x2)
$env \leftarrow AuthEnc_{rw}(c, S)$
$hpw \leftarrow (k, env, s, C)$

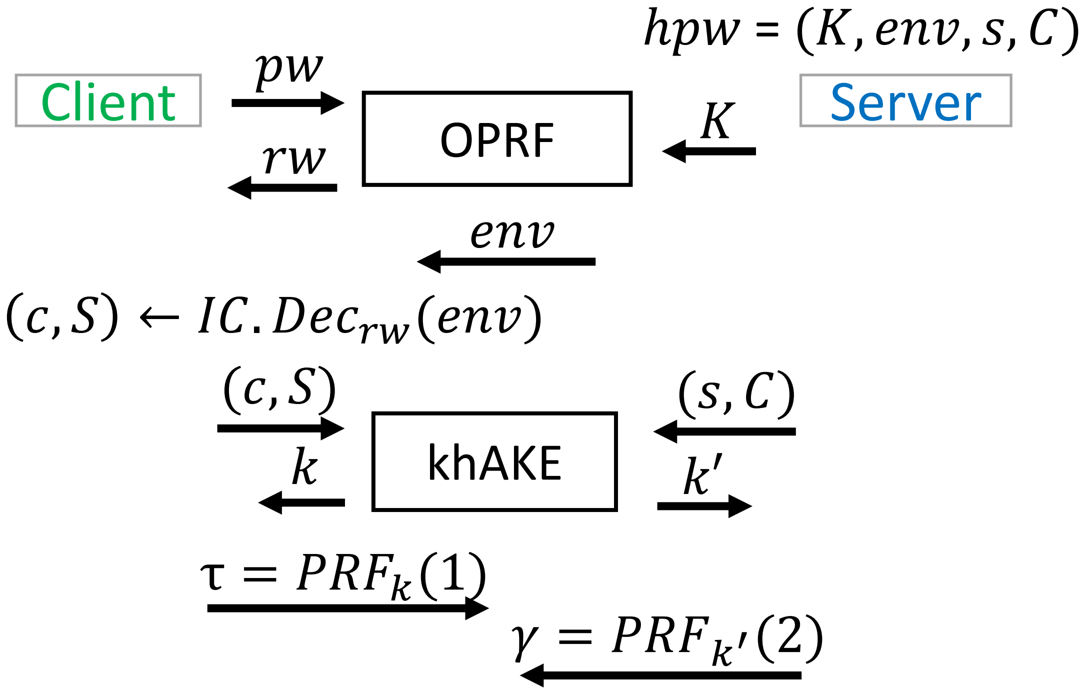
OPAQUE	vs	OPAQUE'
3 flows		4 flows
any* KCI-AKE *e.g. *Sigma* [Kra03] of TLS		khAKE
if (IT)OPRF key leaks: pw leaks		⇒ aPAKE

OPAQUE' [JKX18] *(= OPRF + aPAKE)*

↑ *KHAPE*

$K \leftarrow OPRF.KG$ (oblivious PRF)
$rw \leftarrow OPRF_K(pw)$
$(c, C), (s, S) \leftarrow khAKE.KG$ (x2)
$env \leftarrow IC.Enc_{rw}(c, S)$
$hpw \leftarrow (K, env, s, C)$





**KCI = key compromise impersonation [security]*
≈ perf. forward secrecy ≈ key conf. flows