

Co Low-Entr

based

Michael Rosenberg
Cloudflare Research
US



(PrivCrypt)

Lucjan Hanzlik (CISPA)

Daniel Slamanig (Universität der Bundeswehr München)

Web site of the workshop

Workshop on Quantum-Safe Hybrid Cryptography (QSHC)

22

Ludovic Perret (EPITA France)

Christoph Striecks (Austrian Institute of Technology)

Web site of the workshop

6th ACNS Workshop on Secure Cryptographic Implementation (SCI)

Jingqiang Lin (University of Science and Technology of China)

Bo Luo (University of Kansas)

Web site of the workshop

Workshop on Secure Protocol Implementations in the Quantum Era (SPIQE)

↓ ↓

Kenneth G. Paterson (ETH Zürich)

Juraj Somorovsky (University of Paderborn)

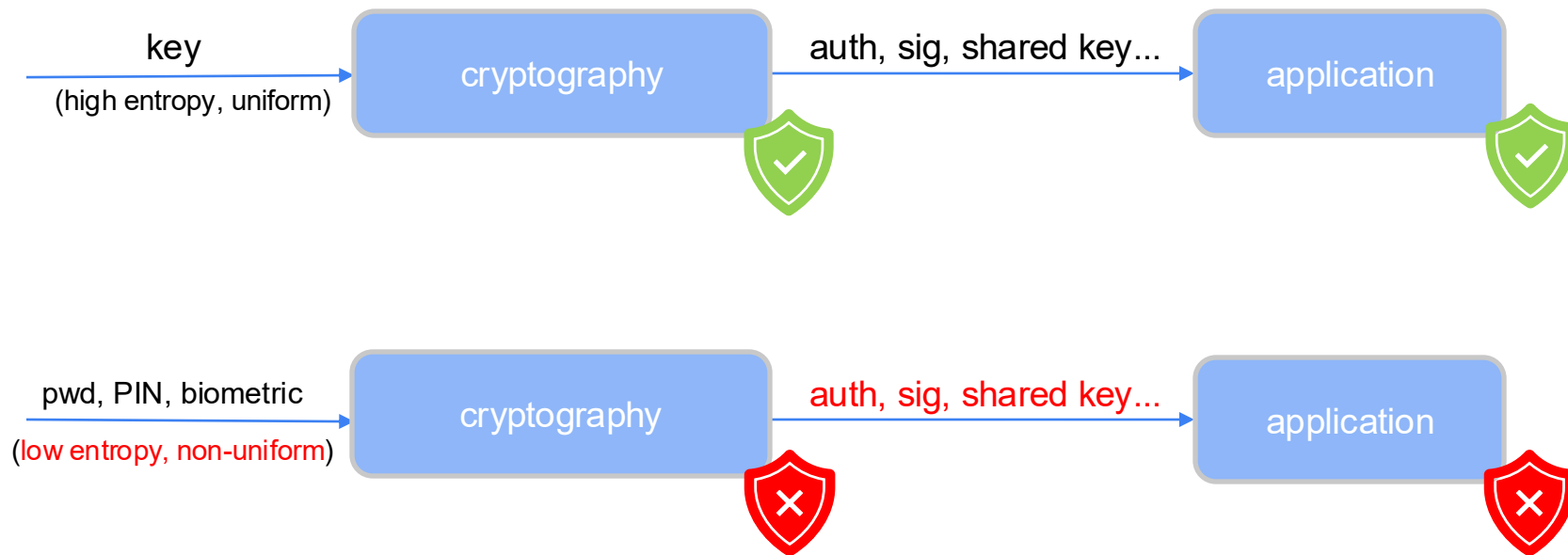
Web site of the workshop

7th ACNS Workshop on Security in Machine Learning and its Applications (SIML)

11



On cryptographic keys

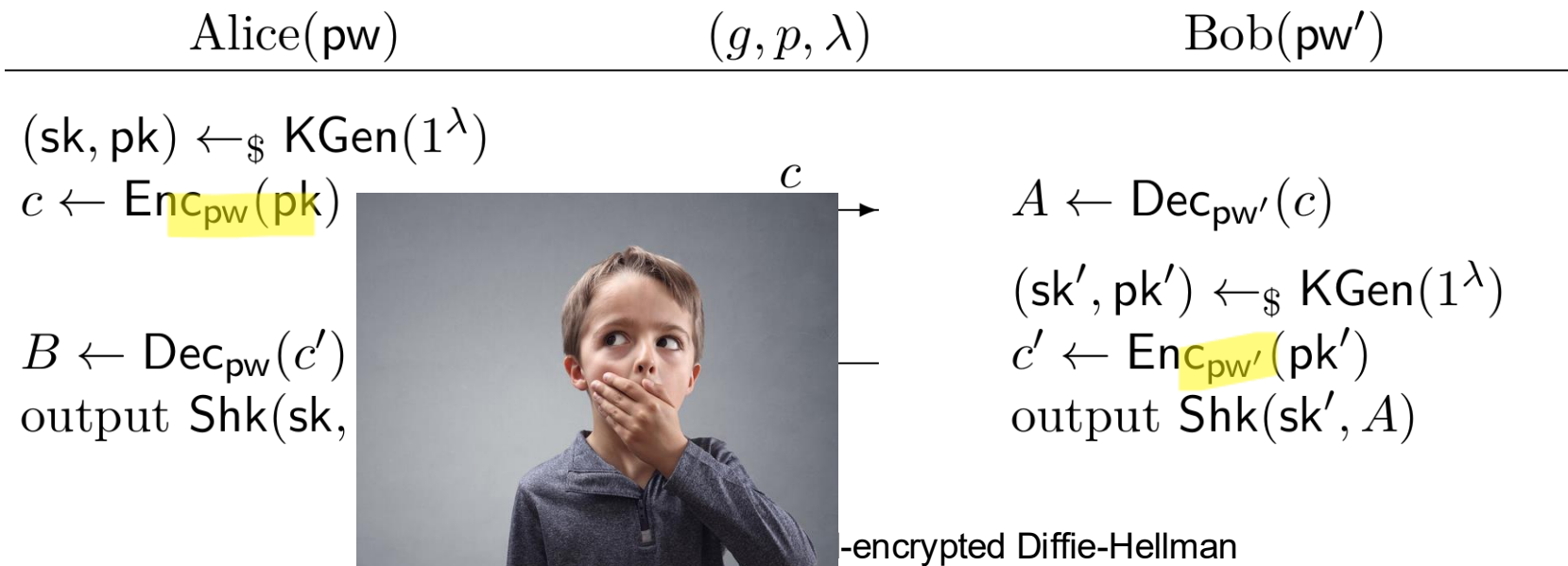


Low-entropy cryptography



Cryptography that works well
with „bad“ secrets/keys

Example: Password-authenticated key exchange



Example: Password-based key exchange

Alice(pw)

$(sk, pk) \leftarrow_{\$} \text{KGen}(1^\lambda)$

$c \leftarrow \text{Enc}_{pw}(pk)$

$B \leftarrow \text{Dec}_{pw}(c')$

output $\text{Shk}(sk, B)$

Bob(pw')

$A \leftarrow \text{Dec}_{pw'}(c)$

$(sk', pk') \leftarrow_{\$} \text{KGen}(1^\lambda)$

$c' \leftarrow \text{Enc}_{pw'}(pk')$

output $\text{Shk}(sk', A)$

EKE [BelMer92]

Diffie-Hellman

Composable security in the Ideal Cipher model [EC:DHPY18, EC:JanRoyXu24]

proof flawed
(Appx. A.1)

result incorrect
(Sects. 3.2 and 3.3,
Appx. A.4)

result incorrect
(Sect. 3.2
and Appx. A.4)
result ambiguous,
unclear if OEKE-PRF
or OEKE-RO either
way result incorrect
(Sect. 3.1)

result incorrect
(Sect. 3.3, Appxs. A.1
and A.3)

result incorrect
(Appx. A.3)

Low-entropy primitives

Password-authenticated key exchange (**PAKE**)

Exchange a symmetric key from a shared password

- Symmetric (both parties share password)
- Asymmetric (server stores encoded password)
- Fuzzy (tolerates errors in password)
- Distributed/threshold (server role shared)

Password-protected secret sharing (**PPSS**)

Share and recover a secret with many servers

- Threshold, fuzzy...

Password-protected key retrieval (**PPKR**)

Like PPSS but rate-limited

- Fuzzy, distributed, threshold,...

Oblivious Pseudo-random Function (**OPRF**)

2-party computation of a PRF

- Allows enhancing password entropy
- Advantage over hashing: server remembers salt, rate-limiting, precomputation protection
- Has become a design paradigm of low-entropy schemes: OPRF + standard crypto

Back in 2018...

Matthew Green in passwords, protocols, provable security · October 19, 2018 · 2,779 Words

Let's talk about PAKE

The first rule of PAKE is: nobody ever wants to talk about PAKE. The second rule of PAKE is that this is a shame, because PAKE — which stands for Password Authenticated Key Exchange — is actually one of the most useful technologies that (almost) never gets used. It should be deployed everywhere, and yet it isn't.

To understand why this is such a damn shame, let's start by describing a very real problem.



Matthew Green

I'm a cryptographer and professor at Johns Hopkins University. I've

Identified 2 main reasons why PAKEs are not used

- (1) There's a lack of good PAKE implementations in useful languages
- (2) Cryptographers are bad at communicating the value of their work

But now, finally...

Password-authenticated key exchange (PAKE)

Exchange a symmetric key from a shared password



Password-protected secret sharing (PPSS)

Share and recover a secret with many servers



Password-protected key retrieval (PPKR)

Like PPSS but rate-limited

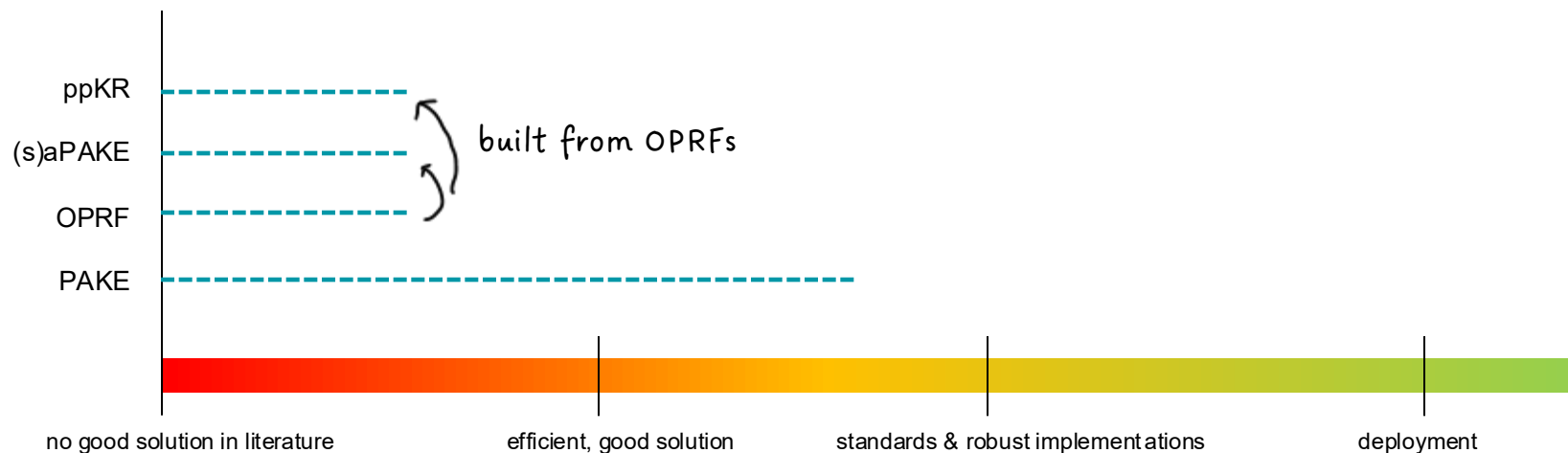


Oblivious Pseudo-random Function (OPRF)

2-party computation of a PRF



Post-quantum low-entropy cryptography



Transition now? Not necessary for authentication...

Authentication

User and machine authentication typically involves the use of a digital signature algorithm or key-establishment scheme. NIST recommends that quantum-vulnerable algorithms **can be used until quantum computers can break them**. At that point, an upgrade will be required. This applies to network security protocols also, where the algorithm used for authentication can be transitioned separately.

Checking login passwords
in a zero-knowledge fashion!

PAKE + key confirmation = **secure password authentication**

Fun fact: this could add password authentication to TLS 1.3 but is not used in practise [EC:HJKW23]

Low-entropy deployments deriving *encryption* keys

Harvest-now-decrypt later attacks

Password-authenticated key exchange (PAKE)

Exchange a symmetric key from a shared password



Password-protected secret sharing (PPSS)

Share and recover a secret with many servers



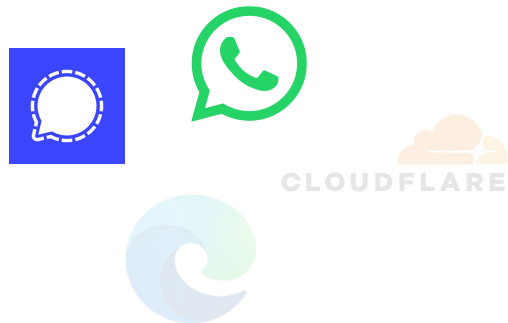
Password-protected key retrieval (PPKR)

Like PPSS but rate-limited

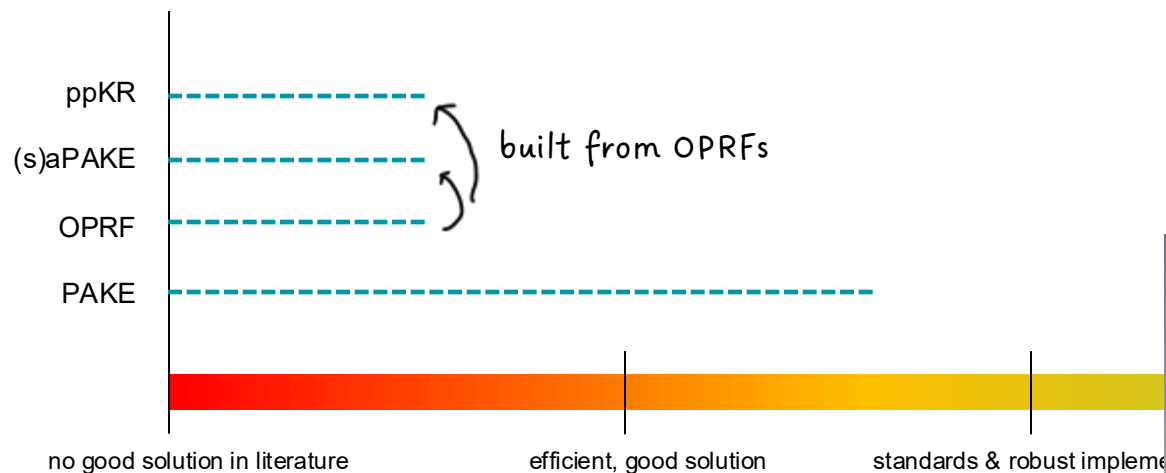


Oblivious Pseudo-random Function (OPRF)

2-party computation of a PRF



So.... we need to transition asap. But to what?




Some time last year...


From: [****@\[big-company\].\[some-country\]](#)
To: [juliahesse2@gmail.com](#)
Subject: Post-quantum OPRFs

Hi Julia,

Legendre-based OPRF,
complex construction



we saw your paper that just came up on eprint, and we were wondering whether it's a good idea to implement it to make our **** deployment post-quantum. Any thoughts?



Millions of
users...

Answer: Please don't!

Some time last year...

From: [****@\[big-company\].\[anonymized-country\]](#)
To: [juliahesse2@gmail.com](#)
Subject: Post-quantum OPRFs

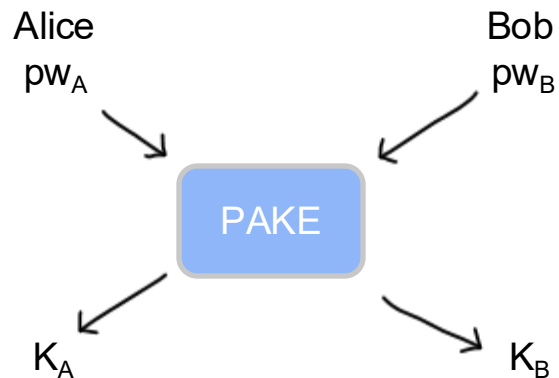
Okay, but what if we hash it together with our
DH-based OPRF?

This talk

Combiners for Low-Entropy Cryptography

Can we build a PAKE/OPRF from a classical and
a pq PAKE/OPRF with just **black-box access**, and
with **best-of-both security**?

Password-Authenticated Key Exchange (PAKE)



$K_A = K_B$ iff $pw_A = pw_B$
otherwise both random

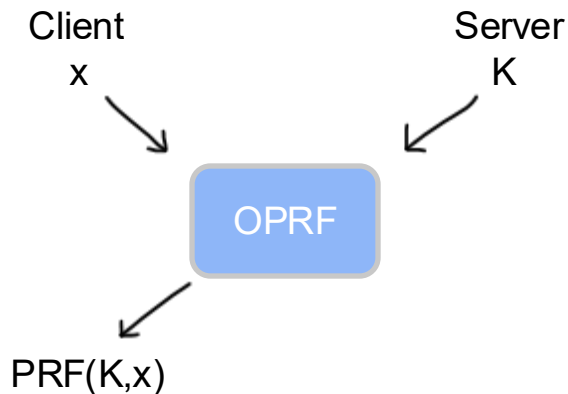
Security properties

- Produces uniform keys
- No offline dictionary attacks on transcript (passive attacker)
- 1 password guess per active attack
- Game-based or simulation-based (composable) notions

~~~~~  
↖ This talk!



# Oblivious Pseudo-Random Function (OPRF)



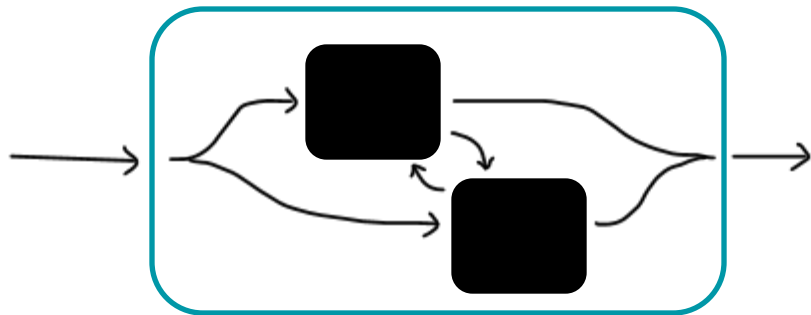
OPRFs are great tools to bootstrap uniform keys from password - just set  $x = \text{pw}$ !

## Security properties

- Server does not learn anything about  $x$
- Client does not learn anything about  $K$  beyond the output
- **Guaranteed uniform outputs** for client (even if server malicious)
- **Game-based or simulation-based (composable) notions**

Allows modular protocol design with OPRFs

# Black-box combiner



Black box property

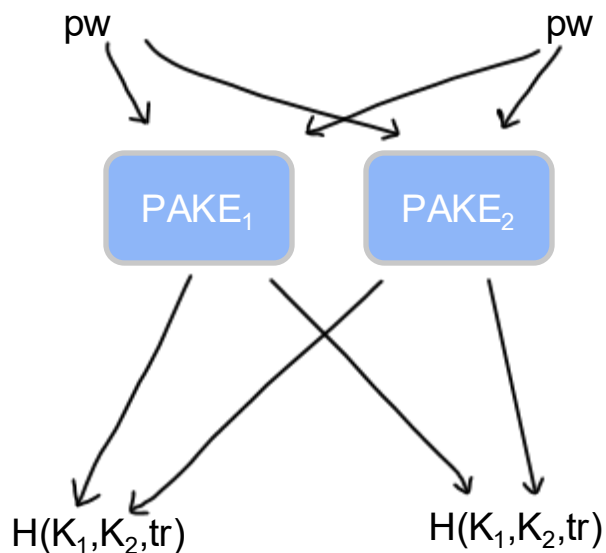
- Black-box access
- A break of combiner always results in the break of one of the building blocks

tldr: no matter how a component breaks, it does not make the whole thing insecure!

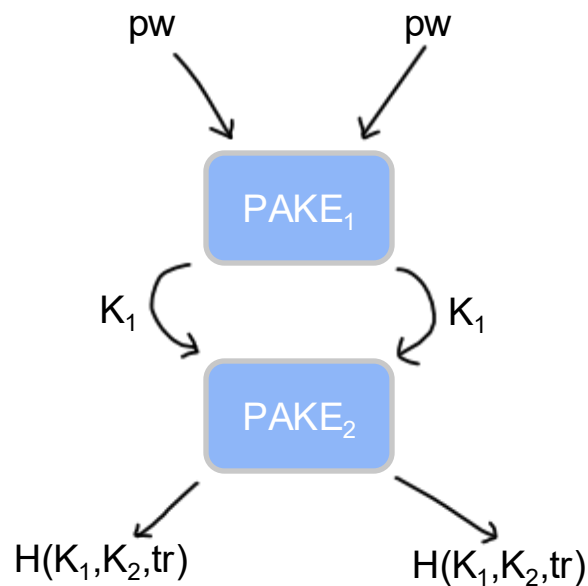
PRF combiner  $\text{PRF}_1(K, x) \oplus \text{PRF}_2(K', x)$

KEM combiner  $H(K_1, K_2, c_1, c_2)$

# Combining PAKEs – natural approaches

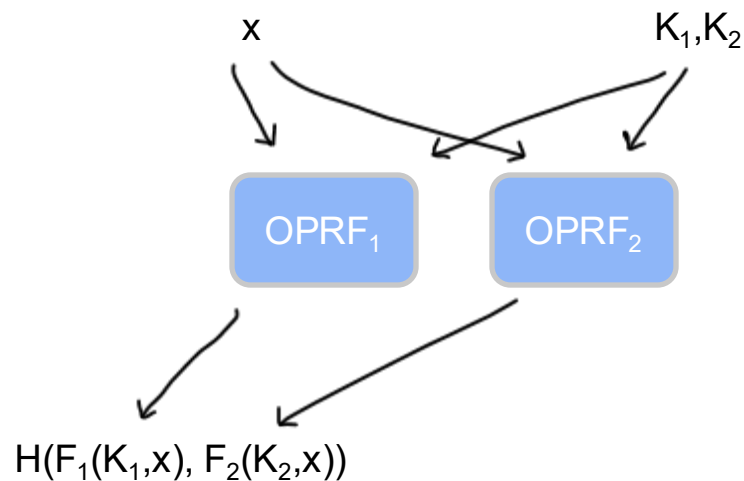


„Parallel“ combiner

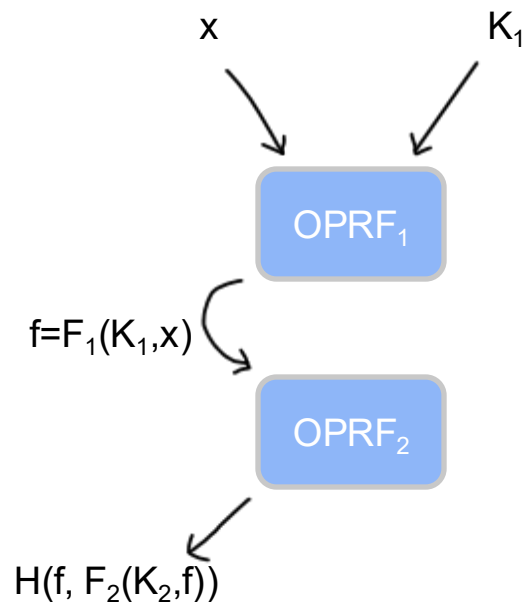


„Sequential“ combiner

# Combining OPRFs – natural approaches



„Parallel“ combiner



„Sequential“ combiner

# Challenges in combining low-entropy cryptography



Components either **cannot break** in such a way that they leak information about their input, OR they **can't be fed** pw/x (???)

# Running into hard problems

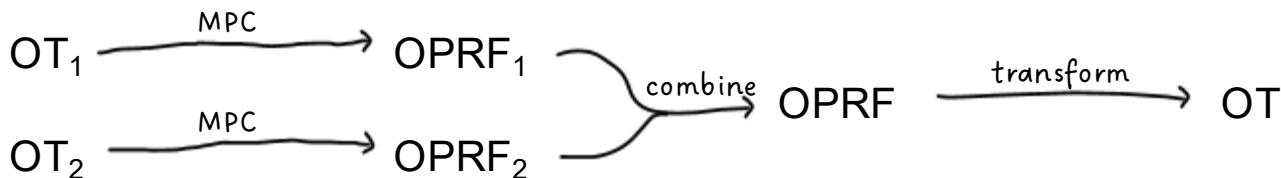
- OPRFs imply OT:

OT sender chooses a PRF key  $K$

OT sender encrypts OT inputs with  $\text{PRF}(K,0)$  and  $\text{PRF}(K,1)$  and sends both ctexts

OT receiver evaluates the PRF at its choice bit, and decrypts one of the ctexts

- With this we can build an OT combiner from an OPRF combiner



# Running into hard problems

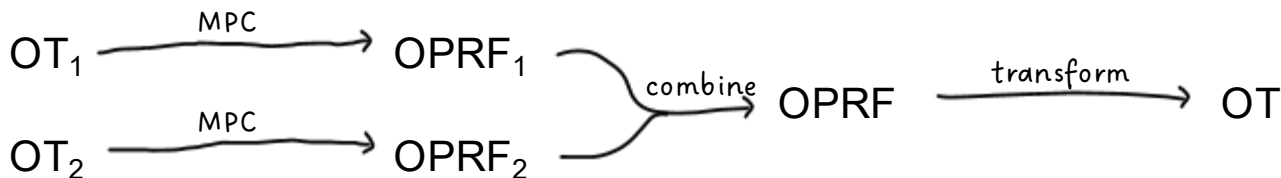
- OPRFs imply OT:

OT sender chooses a PRF key  $K$

OT sender encrypts OT inputs with  $\text{PRF}(K,0)$  and  $\text{PRF}(K,1)$  and sends both ctexts

OT receiver evaluates the PRF at its choice bit, and decrypts one of the ctexts

- With this we can build an OT combiner from an OPRF combiner



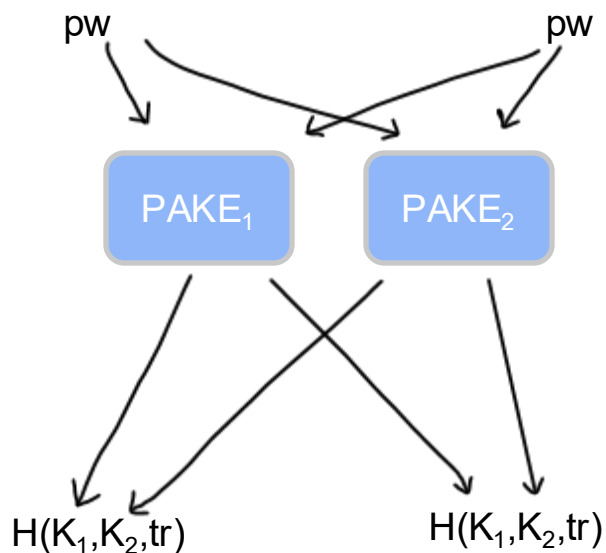
- Impossibility of black-box OT combiner [EC:HKRR05]

indicates that black-box combining OPRFs is hard

Enough theory, let's try to combine some PAKEs!



# Combining PAKEs – in parallel



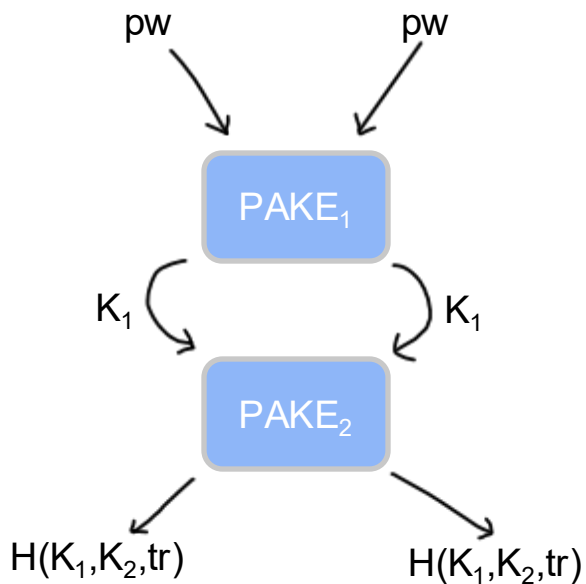
Yields a secure PAKE if **both PAKEs** statistically hide the input passwords

Instantiations:

- EKE, CPace, SPAKE2 (classical)
- None... (post-quantum)

*Not quite there yet!*

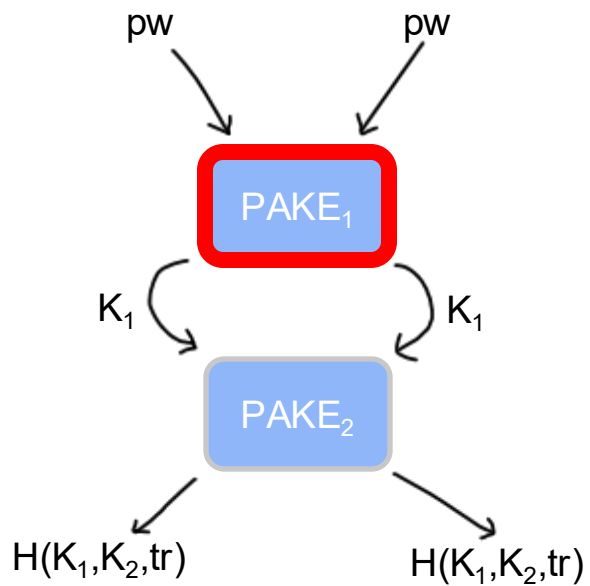
# Combining PAKEs – sequentially



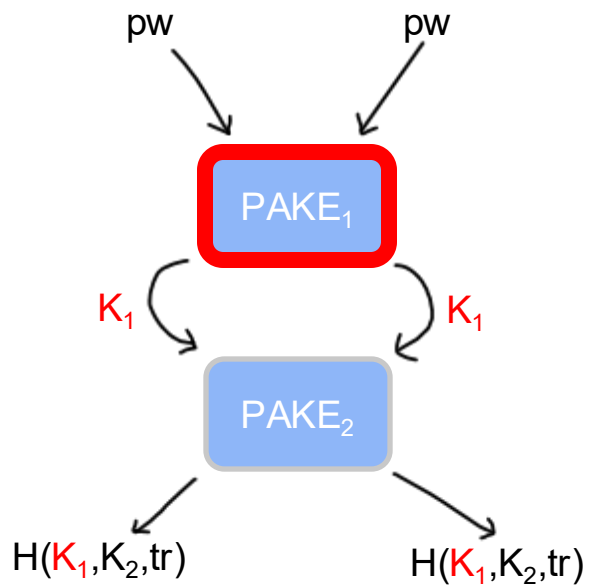
Goal: no statistical input hiding properties on at least one PAKE

Idea:  $K_1$  does not allow brute-force attacks on pw – PAKE<sub>2</sub> can leak  $K_1$

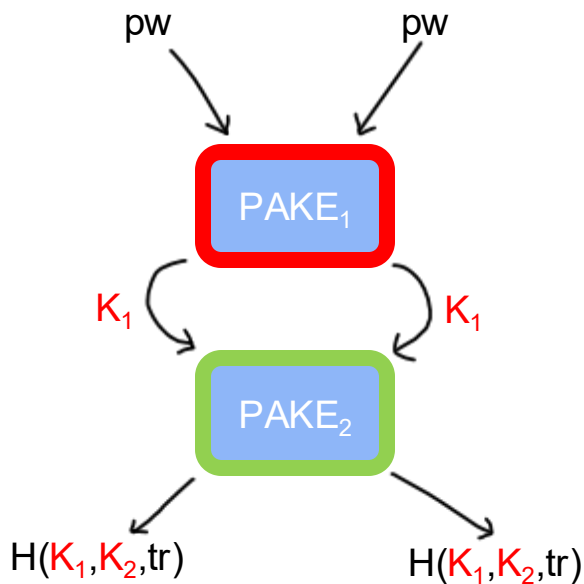
# Combining PAKEs – sequentially



# Combining PAKEs – sequentially



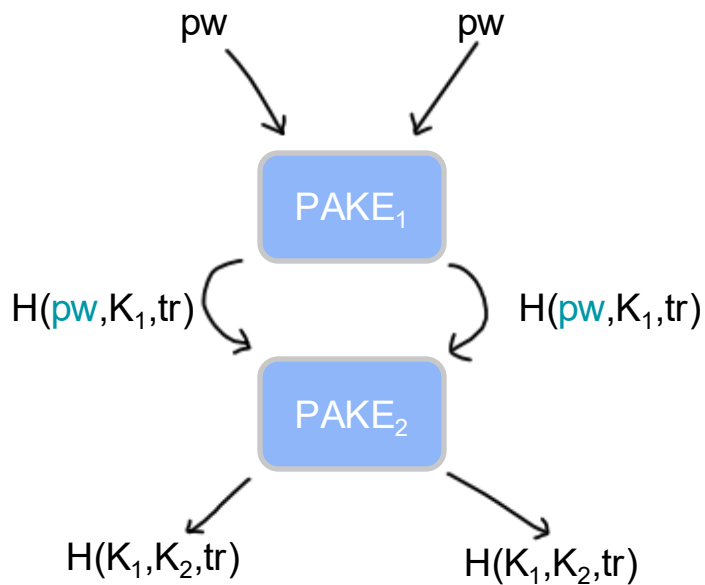
# Combining PAKEs – sequentially



Problem: although second PAKE can be assumed to be secure, it becomes attackable through predicting  $K_1$

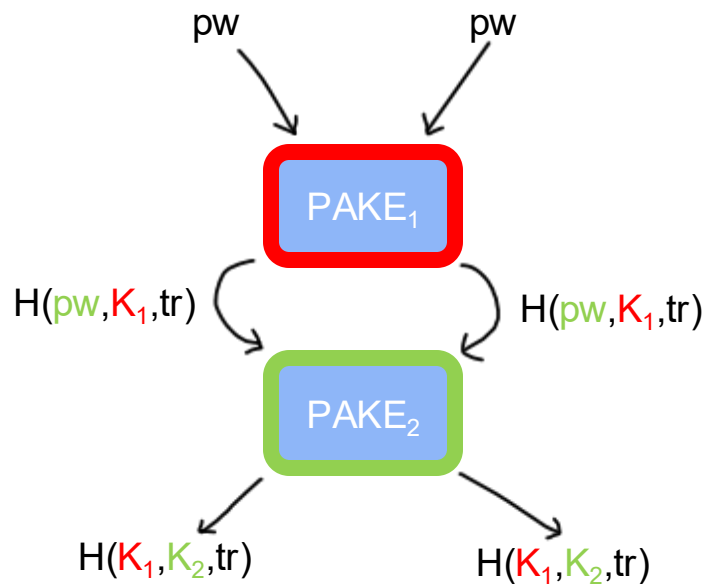
Active adversary can exchange a key without knowing pw

# Combining PAKEs – sequentially



Fix: ensure that attacking  $\text{PAKE}_2$  **implies** a password guess

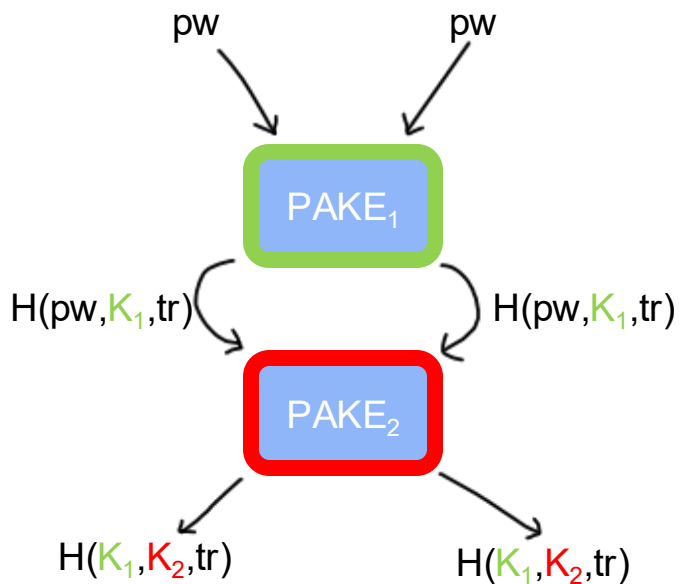
# Combining PAKEs – sequentially: Let's check!



Case PAKE<sub>1</sub> broken:

- PAKE<sub>2</sub> ensures pseudorandomness
- PAKE<sub>1</sub> statistically hides pw

# Combining PAKEs – sequentially: Let's check!

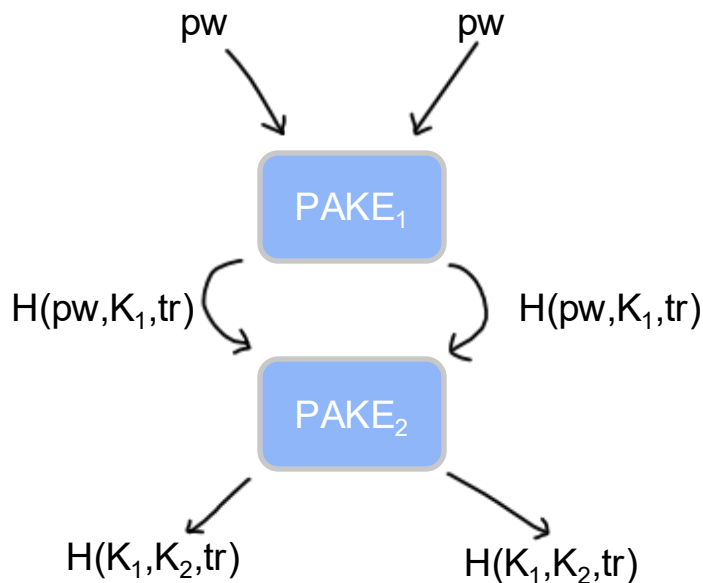


Case PAKE<sub>2</sub> broken:

- PAKE<sub>1</sub> ensures pseudorandomness
- PAKE<sub>2</sub> leaking its input does not expose pw to dictionary attacks thanks to the entropy in  $K_1$
- (Small) caveat: PAKE<sub>2</sub> needs to statistically hide equality of high-entropy inputs



# Combining PAKEs – sequentially



Yields a secure PAKE if PAKE<sub>1</sub> statistically hides the input passwords, and PAKE<sub>2</sub> statistically hides high-entropy input equality

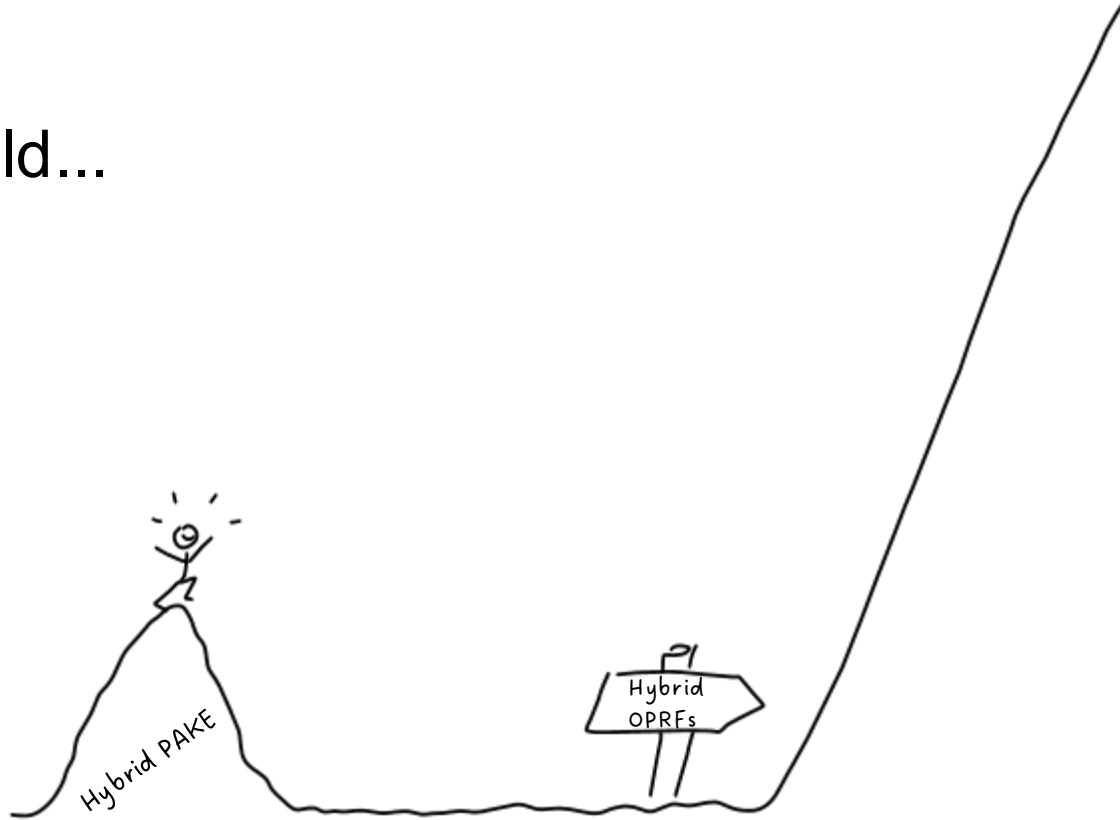
Instantiations

PAKE<sub>1</sub>: EKE, CPace, SPAKE2 (classical)

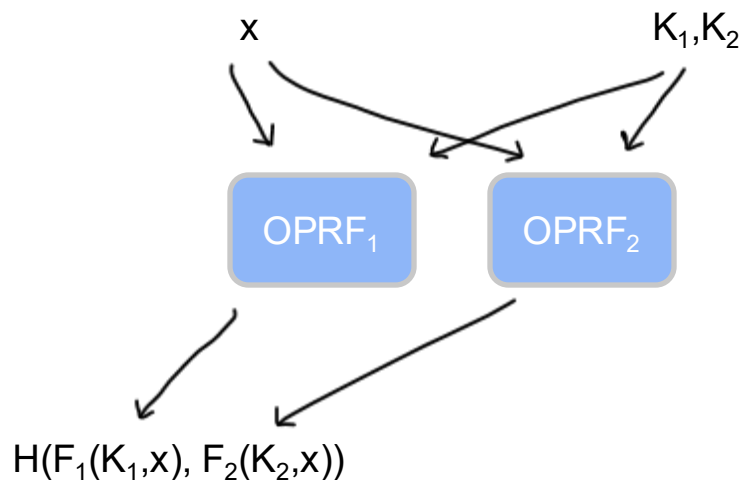
PAKE<sub>2</sub>: OCAKE, CAKE, CHIC (post-quantum)

yay!

Behold...



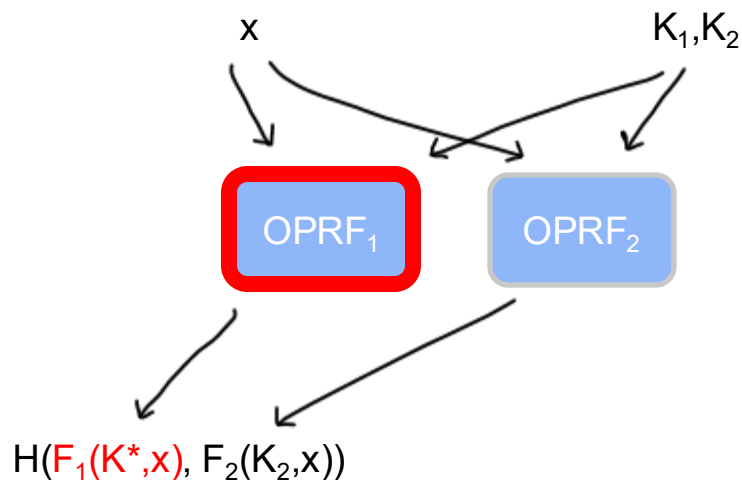
# Combining OPRFs – in parallel



Remember: black-box combiner not feasible, so we need additional assumptions

Hope: with **two statistical input-hiding OPRFs**, this is a secure OPRF even if one of the underlying OPRFs break computationally

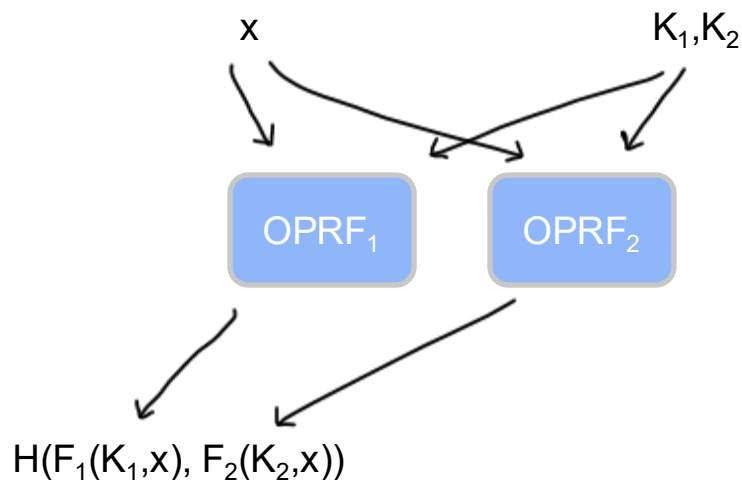
# Combining OPRFs – Statistical input-hiding is not sufficient



Problem: predictable outputs of  $\text{OPRF}_1$  cause simulation failures when extracting keys from active attacks

Violates property of uniform outputs in the presence of active attacker

# Combining OPRFs – Statistical input-hiding is not sufficient



Fix: require **statistical client security**

Instantiations

OPRF<sub>1</sub>: 2HashDH (classical)

OPRF<sub>2</sub>: Legendre-based 2HashPRF (post-quantum)\*, \*\*

\* NOT as implemented in the paper (requires statistical OT instead of computational)

\*\* Unclear efficiency, can be significantly slower as in benchmarks

*Not quite there yet!*

# Responding to that simple question

From: [\\*\\*\\*\\*@\[big-company\].\[anonymized-country\]](#)  
To: [juliahesse2@gmail.com](#)  
Subject: Post-quantum OPRFs

Okay, but what if we hash it together with our DH-based OPRF?

Answer: This can give you a secure OPRF *\*only if\** you implement that post-quantum one with a statistically secure OT, which **decreases efficiency** compared to the paper benchmarks.

Be aware that a **wrongly implemented or theoretically flawed post-quantum OPRF can harm the security of your existing DH-based deployment (!)**

# Take this away

There are **no black-box combiners** for PAKEs, OPRFs, ppKRs,...

All combiners from this talk can lead to *\*insecure\** protocols if the required properties do not hold

Still, they are the **best way we know to transition** to post-quantum low-entropy algorithms

They do **protect against failures in the underlying assumptions**, i.e., a quantum attacker breaking DH, or a flawed post-quantum assumption

## Combiners for PAKE

(with Michael Rosenberg)

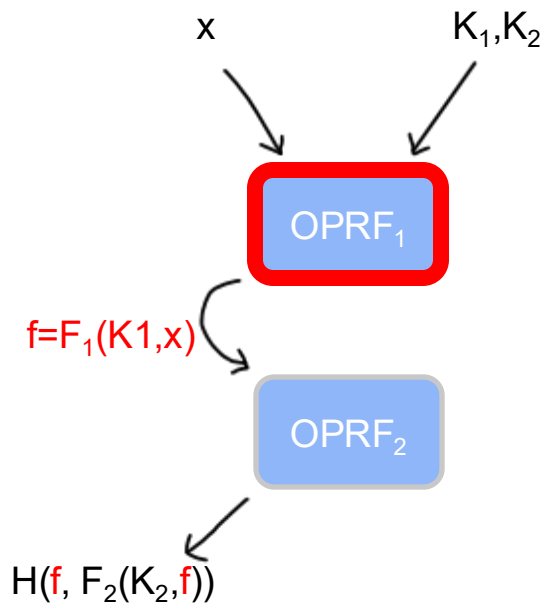
<https://eprint.iacr.org/2024/1621>

## Combiners for OPRFs

(with Sebastian Faller)

<https://eprint.iacr.org/2025/1084>

# Combining OPRFs – Sequential does not help



Intuition: leaking  $F_1(K_1, x)$  and  $K_1$  exposes  $x$  to **offline attacks** if  $K$  is also leaked

*Why does it work in PAKE?*

Sequential PAKE combiner principle:  
build a secure channel with  $\text{PAKE}_1$ , and  
execute  $\text{PAKE}_2$  in it.

We can trust a PAKE party who knows  
the same password as we do. But we  
can **never trust an OPRF server**. So a  
channel to the server does not help us.