How to Tolerate Typos in Strong Asymmetric PAKE

August 20th, 2025

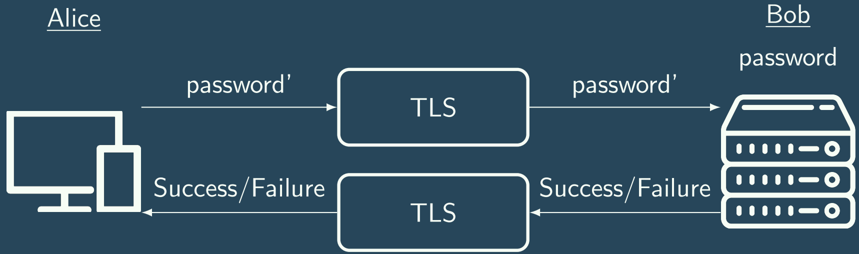
[1]Work done at

[2]Oregon State University

Ian McQuoid[1], Mike Rosulek[2], Jiayu Xu[2]

Auth Problem — Password-over-TLS

- Passwords form the most common method of human authentication online



Assuming secure and server-authenticated channels, password auth is easy!

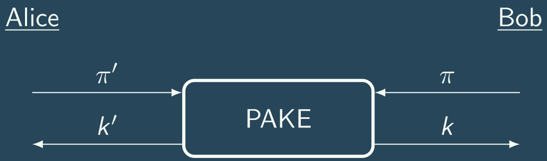
Auth Problem — Goals

Goals

1. Securely derive a key
2. Assuming *only a shared password*
3. Protecting against server compromise
4. And allowing fuzzy matching

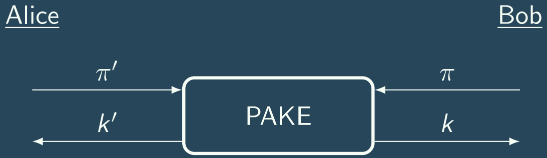
Password-Authenticated Key Exchange [PAKE]

- We can achieve authentication with Password-Authenticated Key Exchange (PAKE)
- When $\pi \neq \pi'$, the participants get independent keys (k, k')



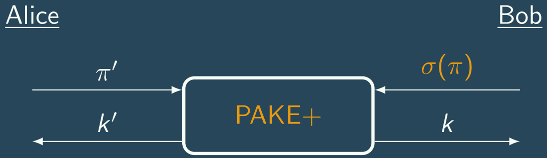
PAKE — Server Compromise

- Server stores π *in the clear*
- What happens if server's storage is stolen?



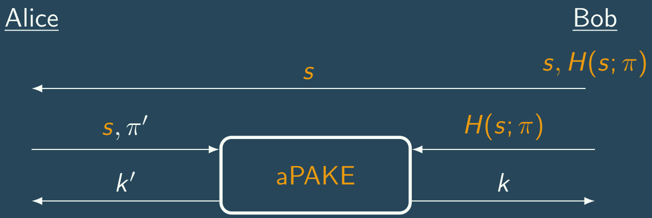
PAKE — Server Compromise

- Instead of π, store some transformation $\sigma(\pi)$
- If $\pi = \pi'$, then $k = k'$ (and independently sampled otherwise)



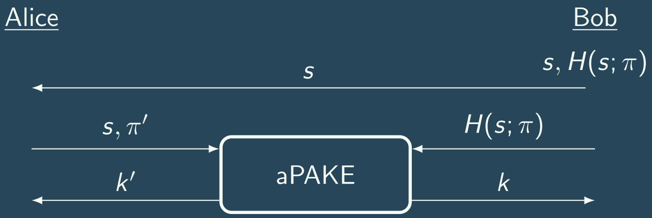
PAKE — Server Compromise

- Bob needs *some* information to authenticate Alice
- As intuition, consider a publicly salted hash $\sigma : \pi \mapsto s, H(s; \pi)$



PAKE — Server Compromise

- Bob needs *some* information to authenticate Alice
- As intuition, consider a publicly salted hash $H(s; \pi)$



Unfortunately, some *non-inevitable* attacks still exist

aPAKE — Precomputation

1. Mallory performs precomputation to help invert the mapping

Lookup Table

Password	Server Storage
π_1	$H(s;\pi_1)$
π_2	$H(s;\pi_2)$
$\vdots$	$\vdots$

aPAKE — Precomputation

1. Mallory performs precomputation to help invert the mapping
2. Mallory steals Bob's storage $H(s;\pi)$
3. Mallory checks her table for $H(s;\pi)$

Lookup Table

Password	Server Storage
π_1	$H(s;\pi_1)$
π_2	$H(s;\pi_2)$
$\vdots$	$\vdots$
π_i	$H(s;\pi_i) = H(s;\pi)$
$\vdots$	$\vdots$

aPAKE — Precomputation

1. Mallory performs precomputation to help invert the mapping
2. Mallory steals Bob's storage $H(s;\pi)$
3. Mallory checks her table for $H(s;\pi)$

Lookup Table

Password	Server Storage
π_1	$H(s;\pi_1)$
π_2	$H(s;\pi_2)$
$\vdots$	$\vdots$
π_i	$H(s;\pi_i) = H(s;\pi)$
$\vdots$	$\vdots$

Can we stop Mallory from precomputing H?

Auth Problem — Goals

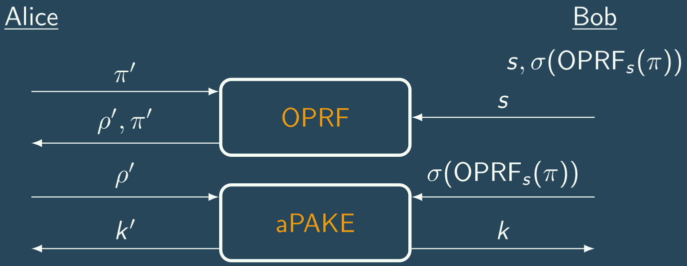
- Can we interactively evaluate $H(s; \pi)$ and hide s?

Goals

1. Securely derive a key
2. Assuming *only a shared password*
3. Protecting against server compromise
4. And allowing fuzzy matching

saPAKE — [JKX18]

- *Strong* aPAKE [JKX18] addresses the precomputation problem.
- Add an interactive step to aPAKE: Oblivious Psuedorandom Functions [OPRF].


Alice
Bob
$s, \sigma(\mathsf{OPRF}_s(\pi))$
π'
OPRF
s
ρ', π'
ρ'
$\sigma(\mathsf{OPRF}_s(\pi))$
aPAKE
k'
k

saPAKE

- As intuition for the different PAKE classes:

	Storage	Post-compromise Effort
PAKE	Plaintext Password	$O(1)$
aPAKE	Publicly Salted Hash	$O(\log \lvert\text{Dictionary}\rvert)$
saPAKE	Privately Salted Hash	$O(\lvert\text{Dictionary}\rvert)$

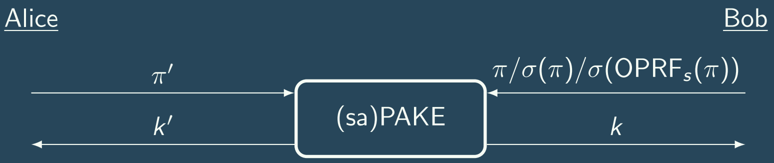
(sa)PAKE — Point Equality

- saPAKE provides most of our requirements
    - Securely derive a key
    - Assuming *only a shared password*
    - Protecting against server compromise

(sa)PAKE — Point Equality

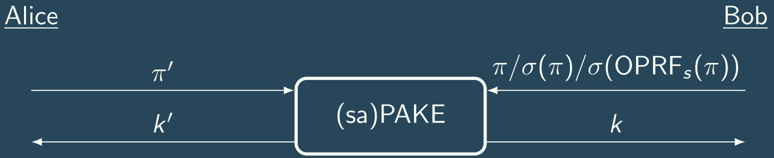
- saPAKE provides most of our requirements
  - Securely derive a key
  - Assuming *only a shared password*
  - Protecting against server compromise
- but (sa)PAKE authenticates a very specific function: *point equality*

(sa)PAKE — Point Equality



$k = k'$ exactly when $\pi = \pi'$

(sa)PAKE — Point Equality

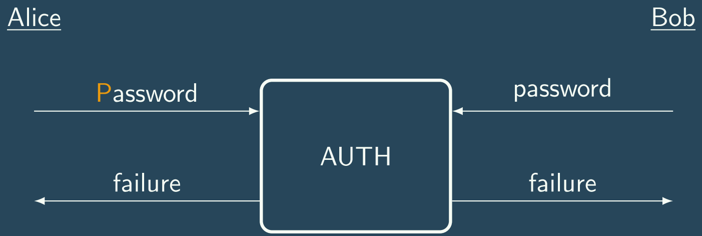


$k = k'$ exactly when $\pi = \pi'$

Can we extend saPAKE to handle typos as well?

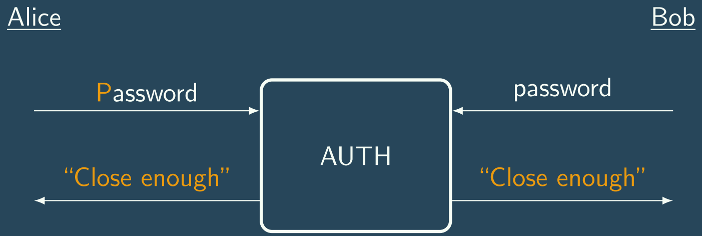
Authentication — Handling Typos

- Inputs can be quite errory
- Users frequently fail to auth with “close” inputs [Cha+16]



Authentication — Handling Typos

- Inputs can be quite errory
- Users frequently fail to auth with “close” inputs [Cha+16]
- Replace (sa)PAKE point functions with fuzzy matching



Authentication — Handling Typos

Typo Policies

Typo Policy	Example
Case-reversal	Password ⇝ pASSWORD
First Case	Password ⇝ password
Repeated First/Last	Password ⇝ PPassword
Adjacent Substitutions	Password ⇝ Padaword

Authentication — Handling Typos

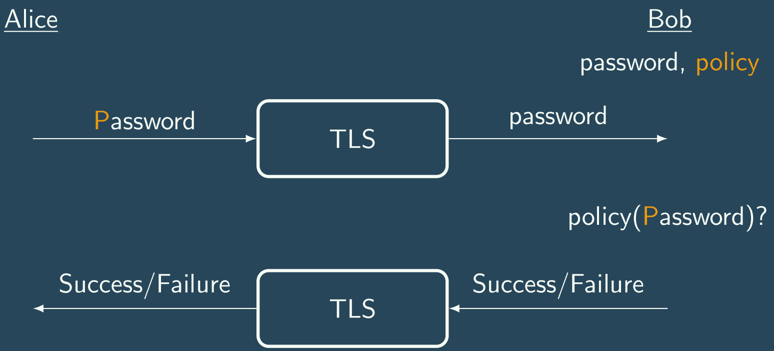
Typo Policies

Typo Policy	Example
Case-reversal	Password ⇝ pASSWORD
First Case	Password ⇝ password
Repeated First/Last	Password ⇝ PPassword
Adjacent Substitutions	Password ⇝ Padaword

Facebook corrects the first three classes

(sa)PAKE — Handling Typos

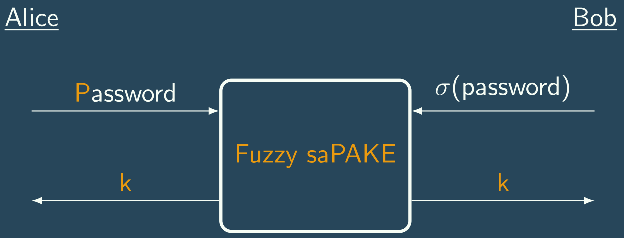
- Fuzzy matching easily addressed in Password-over-TLS



Still unclear how to fuzzy match with compromise resilience

(sa)PAKE — Handling Typos

- RQ: Is there a UC-secure saPAKE with fuzzy matching?



(sa)PAKE — Handling Typos

Previous Work on Universal Composibility Constructions

	Strict Equality	Handles Typos
No Server Compromise	PAKE [Can+05]	fPAKE [Dup+18]
Weak Server Compromise	aPAKE [GMR06]	afPAKE [Erw+20]
Strong Server Compromise	saPAKE [JKX18]	safPAKE

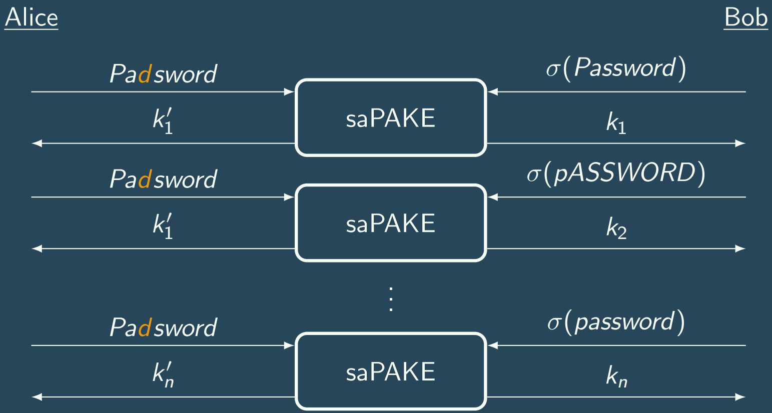
Auth Problem — Goals

Goals

1. Securely derive a key
2. Assuming *only a shared password*
3. Protecting against server compromise
4. And allowing fuzzy matching

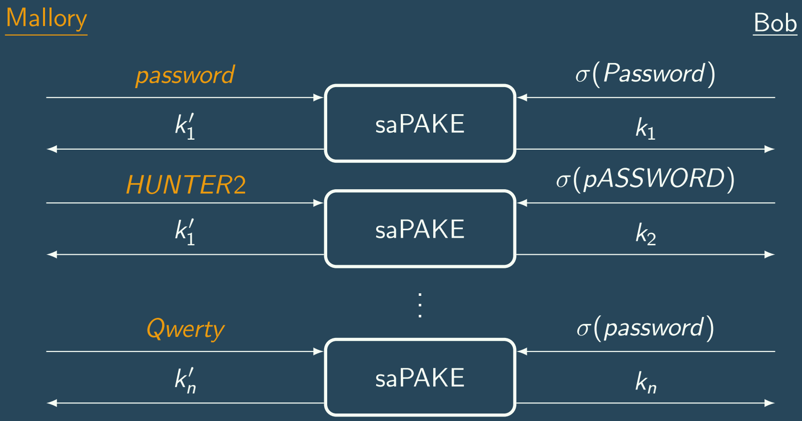
Handling Typos — Naïve Approach

- Run a subsession for each possible typo



Handling Typos — Naïve Approach

- No mechanism to force the *same client password* in each subsession

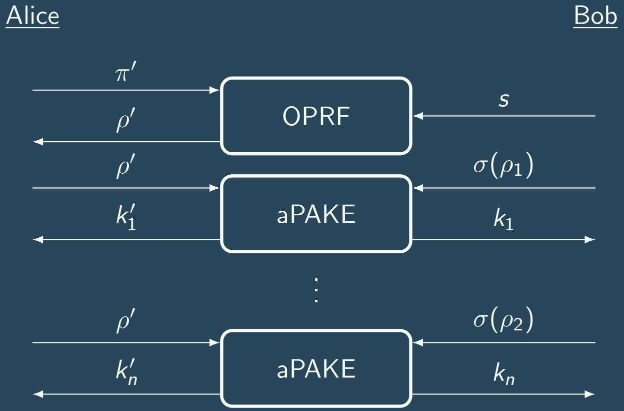


Handling Typos — Naïve Approach

- No mechanism to force the *same client password* in each subsession
- Without OPRF outputs, multiple guesses impossible[1]
- Try compressing all OPRFs into one

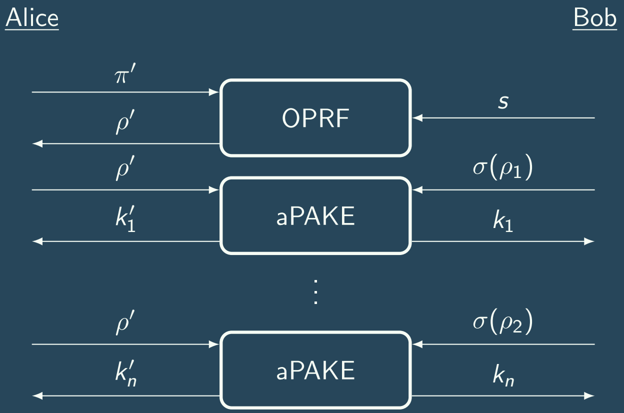
[1] Previous guesses can still be leveraged

Handling Typos — Intuition



- Problem 1: Server can make *unstructured guesses*

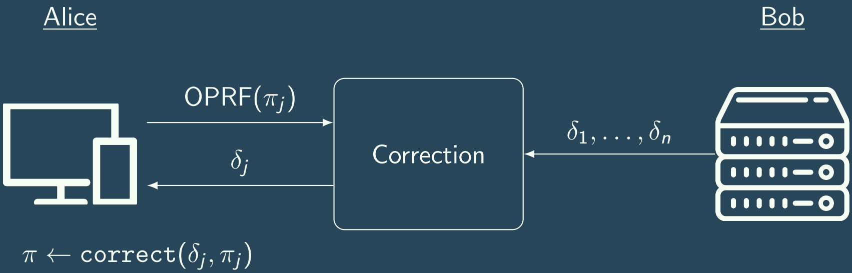
Handling Typos — Intuition



- Problem 1: Server can make *unstructured guesses*
- Problem 2: Communication is still linear in the typo set

Handling Typos — Normalizing Passwords

- Tell Alice *how* to correct her typo



Alice

Bob

OPRF(π_j)

δ_j

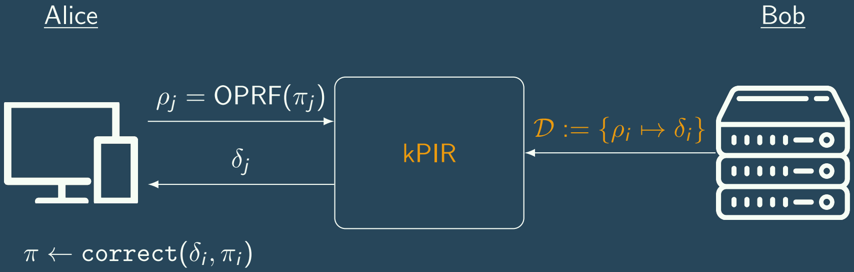
Correction

$\delta_1, \ldots, \delta_n$

$\pi \leftarrow \texttt{correct}(\delta_j, \pi_j)$

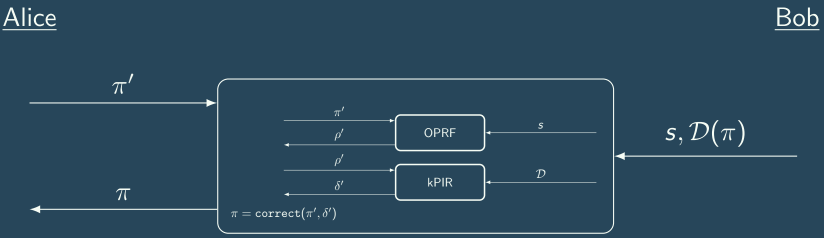
Handling Typos — Normalizing Passwords

- Tell Alice *how* to correct her typo
- Keyword Private Information Retrieval (kPIR) can help!


Alice
Bob
$\rho_j = \mathsf{OPRF}(\pi_j)$
δ_j
kPIR
$\mathcal{D} := \{\rho_i \mapsto \delta_i\}$
$\pi \leftarrow \texttt{correct}(\delta_i, \pi_i)$

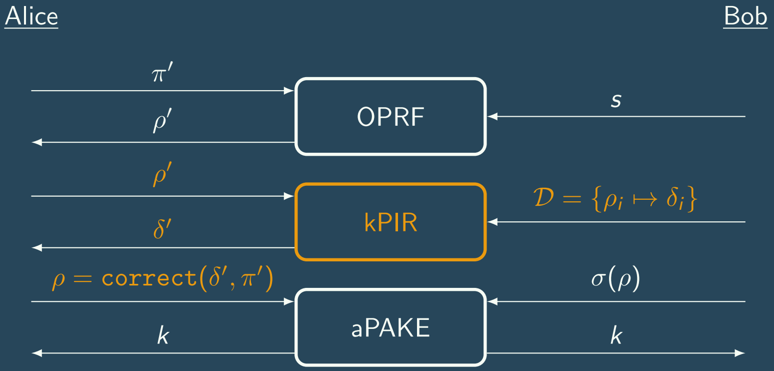
Handling Typos — Normalizing Passwords

- Individually, a protocol obliviously “normalizing” typos



Handling Typos — Normalizing Passwords

- We can compress our n aPAKE steps into one "normalization-then-aPAKE".



Alice
Bob
π'
ρ'
OPRF
s
ρ'
δ'
kPIR
$\mathcal{D} = \{\rho_i \mapsto \delta_i\}$
$\rho = \texttt{correct}(\delta', \pi')$
$\sigma(\rho)$
aPAKE
k
k

Handling Typos — Verifying the Database

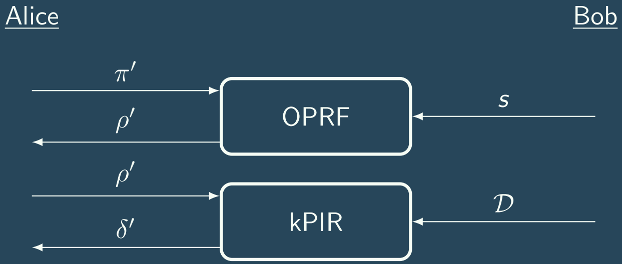
- No mechanism for enforcing an honest database
- Adversary can make independent guesses

Handling Typos — Verifying the Database

- No mechanism for enforcing an honest database
- Adversary can make independent guesses
- Need to prove the server used an expected database!

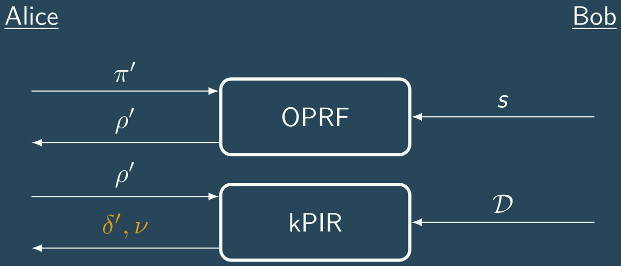
Handling Typos — Verifying the Database

- How do we verify the server acted honestly?



Handling Typos — Verifying the Database

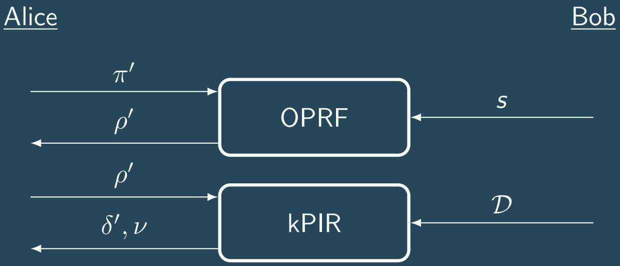
- How could we verify the server acted honestly?
- Client regenerates the server's messages a la Fujisaki-Okamoto



$$\texttt{verify}(\nu) \stackrel{?}{=} (\mathcal{D}, \pi)$$

Handling Typos — Verifying the Database

- How could we verify the server acted honestly?
- Client regenerates the server's messages a la Fujisaki-Okamoto

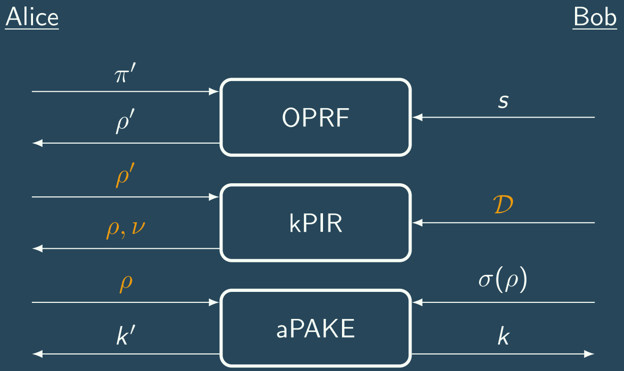


$$\texttt{verify}(\nu) \overset{?}{=} (\mathcal{D}, \pi)$$

Explicitly, ν is an encryption of the random coins necessary to generate $\mathcal{D}$

Handling Typos — safPAKE

- With the proof, safely normalize Alice's typo[2]

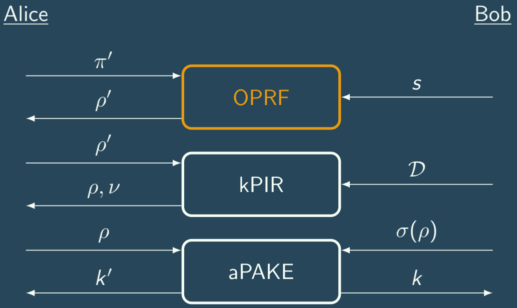


Technically, we also normalize the OPRF output: $\rho' \mapsto \rho$.

Putting Everything Together

safPAKE

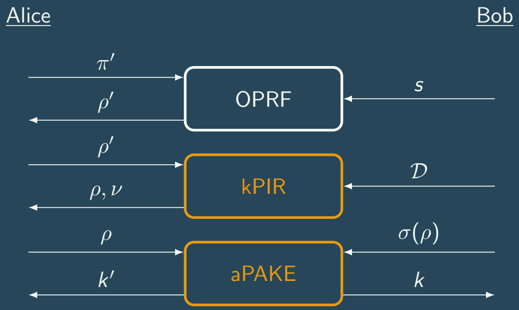
1. Reuse OPRF subsession — Avoid client attacks
2. Normalize aPAKE subsessions — Avoid linear comp/comm costs
3. Verify server's kPIR messages — Avoid server attacks



Putting Everything Together

safPAKE

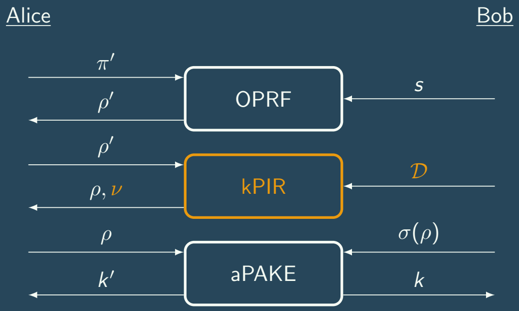
1. Reuse OPRF subsession — Avoid client attacks
2. Normalize aPAKE subsessions — Avoid linear comp/comm costs
3. Verify server's kPIR messages — Avoid server attacks



Putting Everything Together

safPAKE

1. Reuse OPRF subsession — Avoid client attacks
2. Normalize aPAKE subsessions — Avoid linear comp/comm costs
3. Verify server's kPIR messages — Avoid server attacks



Auth Problem — Goals

Goals

1. Securely derive a key
2. Assuming *only a shared password*
3. Protecting against server compromise
4. And allowing fuzzy matching

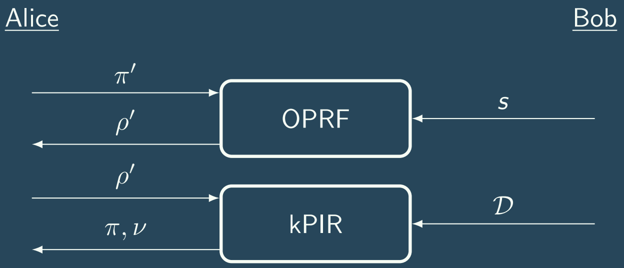
Authentication — Handling Typos

Previous Work on Universal Compposibility Constructions

	Strict Equality	Handles Typos
No Server Compromise	PAKE [Can+05]	fPAKE [Dup+18]
Weak Server Compromise	aPAKE [GMR06]	afPAKE [Erw+20]
Strong Server Compromise	saPAKE [JKX18]	safPAKE (Our Result)

Next Steps

- The normalization step has strong compromise guarantees
- Future typo-tolerant password-based protocols are possible



Next Steps

- Our protocol supports a general notion of similarity
- But the client's computation is linear in the database size
- Leveraging succinct proofs or kPIR optimization allows for larger fuzzy sets

Costs

	Ours + Trivial vPIR	Ours + FHE vPIR
$\mathcal{C}$ Cost	$(n+4)$E, $(n+3)$H	$(n+4)$E, $(n+3)$H, $O(n)$F
$\mathcal{S}$ Cost	3E, 2H	3E, 2H, nF
Rounds	3	5
Communication	$(3n+9)\kappa + 4\mathbb{G}$	$O(\kappa) + 4\mathbb{G} + 9\kappa$
$\mathcal{S}$ Storage	$(3n+1)\kappa$	$O(\kappa)n + (3n+1)\kappa$

Thank You

References

[Can+05] Ran Canetti et al. “Universally composable password-based key exchange”. In: **EUROCRYPT**. 2005.

[Cha+16] Rahul Chatterjee et al. “pASSWORD tYPOS and how to correct them securely”. In: **IEEE Security and Privacy**. 2016.

[Dup+18] Pierre-Alain Dupont et al. “Fuzzy password-authenticated key exchange”. In: **EUROCRYPT**. 2018.

[Erw+20] Andreas Erwig et al. “Fuzzy asymmetric password-authenticated key exchange”. In: **ASIACRYPT**. 2020.

[GMR06] Craig Gentry, Philip MacKenzie, and Zulfikar Ramzan. “A method for making password-based key exchange resilient to server compromise”. In: **CRYPTO**. 2006.

References

[JKX18] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. "OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks". In: **EUROCRYPT**. 2018.