Fantastic OPRFs and where to find them

stf

<2023-07-15 Sat>

Overview

1. theory, mostly based on the awesome SoK: OPRFs[1], by Casacuberta, Hesse & Lehmann
2. practical examples

[1] https://eprint.iacr.org/2022/302

Oblivious Pseudo Random Functions (OPRF)

A building block for privacy.

- oblivious keyword search (KS)
- private information retrieval (PIR)
- private set intersection (PSI)
- password-protected secret sharing (PPSS/TPASS)
- password-authenticated key exchange (PAKE)
- single sign-on (SSO) with privacy
- cloud key management
- de-duplication systems
- secure pattern matching, and
- "untraceable" contact tracing

Step-by-step

- what is an F?
- what is an RF?
- what is a PRF?
- and finally! what is an OPRF?

Functions (F)

$$f : \{0,1\}^m \rightarrow \{0,1\}^n$$

- Definition: A mapping between input values and output values.
- Not to be confused with a subroutine.
- Most of the time algorithmic, sometimes only mapping tables.

Random Functions (RF)

$$f : \{0,1\}^m \xrightarrow{\mathrm{R}} \{0,1\}^n$$

- Function where input values are randomly mapped to output values.
- Theoretical: "too big to store, too slow to compute" [2, 3]
- Random Oracle Model [4, 5, 6]

---

[2] https://blog.cryptographyengineering.com/2011/09/29/what-is-random-oracle-model-and-why-a3/

[3] https://blog.cryptographyengineering.com/2011/10/08/what-is-random-oracle-model-and-why-2/

[4] https://blog.cryptographyengineering.com/2011/10/20/what-is-random-oracle-model-and-why_20/

[5] https://blog.cryptographyengineering.com/2011/11/02/what-is-random-oracle-model-and-why/

[6] https://blog.cryptographyengineering.com/2020/01/05/what-is-the-random-oracle-model-and-why-should-you-care-part-5/

Pseudo Random Functions (PRF)[8]

$$f : \{0,1\}^k \times \{0,1\}^m \xrightarrow{\mathrm{R}} \{0,1\}^n$$

- Goldreich, Goldwasser and Micali ('86)
- Definition: A function that can be used to generate output from a random seed and a data variable, such that the output is computationally indistinguishable from truly random output.[7]
- Like a keyed-hash or MAC (HMAC)

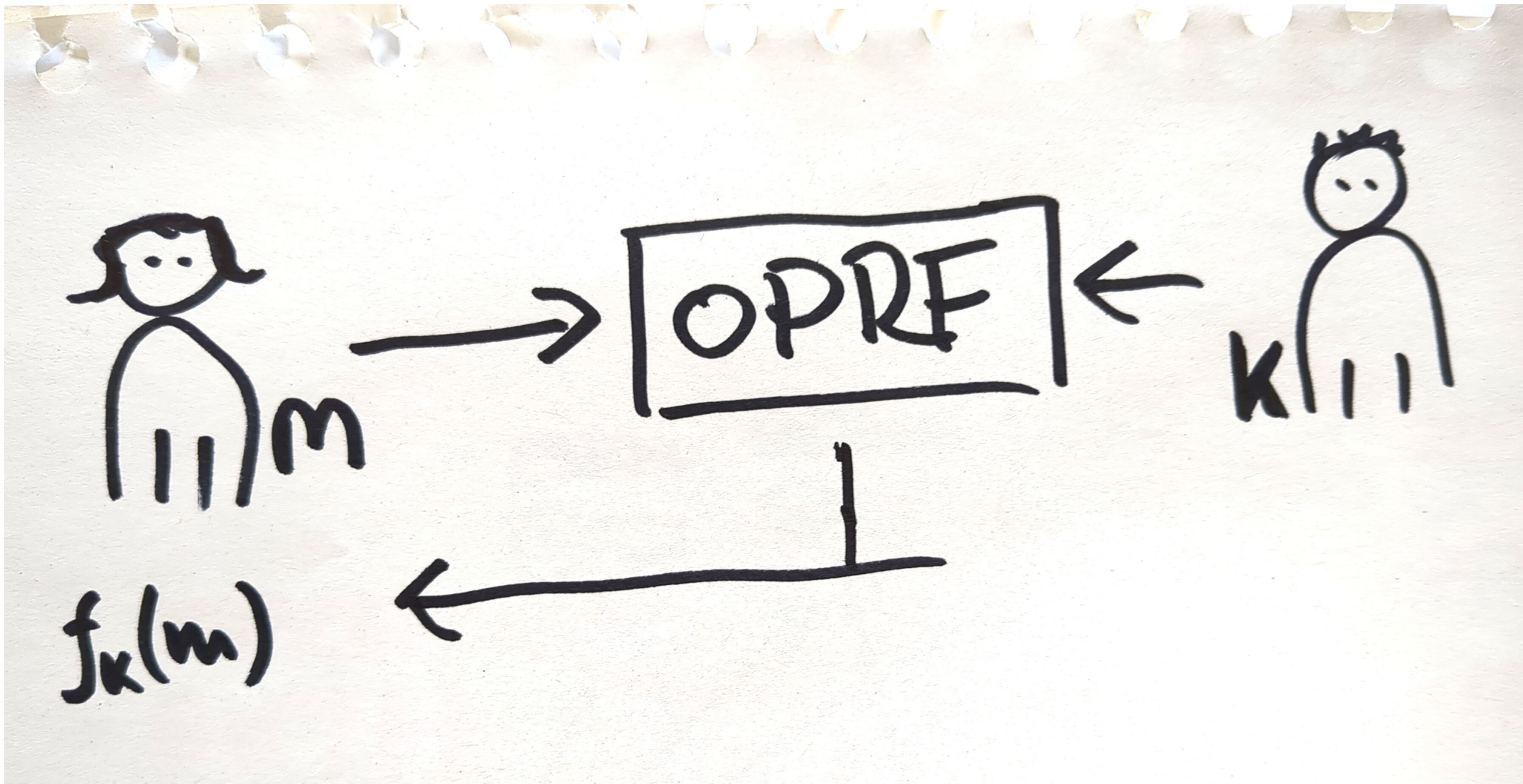
[7] https://csrc.nist.gov/glossary/term/pseudorandom_function

[8] https://www.cs.umd.edu/~jkatz/crypto/f02/lectures/lecture15.pdf

Oblivious Pseudo Random Functions (OPRF)

Client has m, server has k:

$$f : \{0,1\}^k \times \{0,1\}^m \xrightarrow{R} \{0,1\}^n$$



Client learns the result, server learns nothing.

Like an interactive/online/MPC keyed hash/MAC.

OPRF techniques

- Naor and Reingold ('04) + OT/HE
- Dodis-Yampolskiy PRF + HE
- OT/MPC
- (2)HashDH

Naor-Reingold PRF ('04)

$$k := (a_0, ...a_n) \xleftarrow{R} \mathbb{Z}_q^{n+1}$$

$$f_k(m) := g^{a_0 \cdot \prod_{i=1}^{n} a_i^{m_i}}$$

add some Oblivious Transport (OT) or Homomorphic Encryption (HE), shake and voilá: an OPRF

WTH: OT?

> *In cryptography, an oblivious transfer (OT) protocol is a type of protocol in which a sender transfers one of potentially many pieces of information to a receiver, but remains oblivious as to what piece (if any) has been transferred.*[9]

$\binom{1}{2}OT^k$ - send 2 k-bit messages, only one is delivered, sender has no clue which.

[9] https://en.wikipedia.org/wiki/Oblivious_transfer

OT from an OPRF!

1. Alice has m_0 and m_1, and a random key k, she calculates
   $c_0 \leftarrow m_0 \oplus f_k(0)$
   $c_1 \leftarrow m_1 \oplus f_k(1)$
   sends c_0 and c_1 to Bob.
2. Alice and Bob execute an OPRF protocol, from Alice with input k, and Bob with input $b \in 0, 1$: Bob learns: $f_k(b)$
3. Bob can then decrypt $m_b \leftarrow c_b \oplus f_k(b)$ and thus learns m_b, but not the other message.

Naor-Reingold/OT OPRF ('04)

$C(x)$		$S(k)$
		$(r_1, \ldots, r_n) \leftarrow_{\mathrm{R}} \mathbb{Z}_p^n$
		$(a_0, \ldots, a_n) := k$
	$\hat{g}$ ←	$\hat{g} \leftarrow g^{a_0} \prod_{i=1}^{n} 1/r_i$
$x_i := i$-th bit of x		
(for $i = 1, \ldots, n$) do:		
x_i →	$\binom{2}{1}$-OT	← $(r_i, a_i r_i)$
← $r_i a_i^{x_i}$		
(end for)		
$A \leftarrow \prod_{i=1}^{n} r_i a_i^{x_i}$		
Output $\hat{g}^A$		

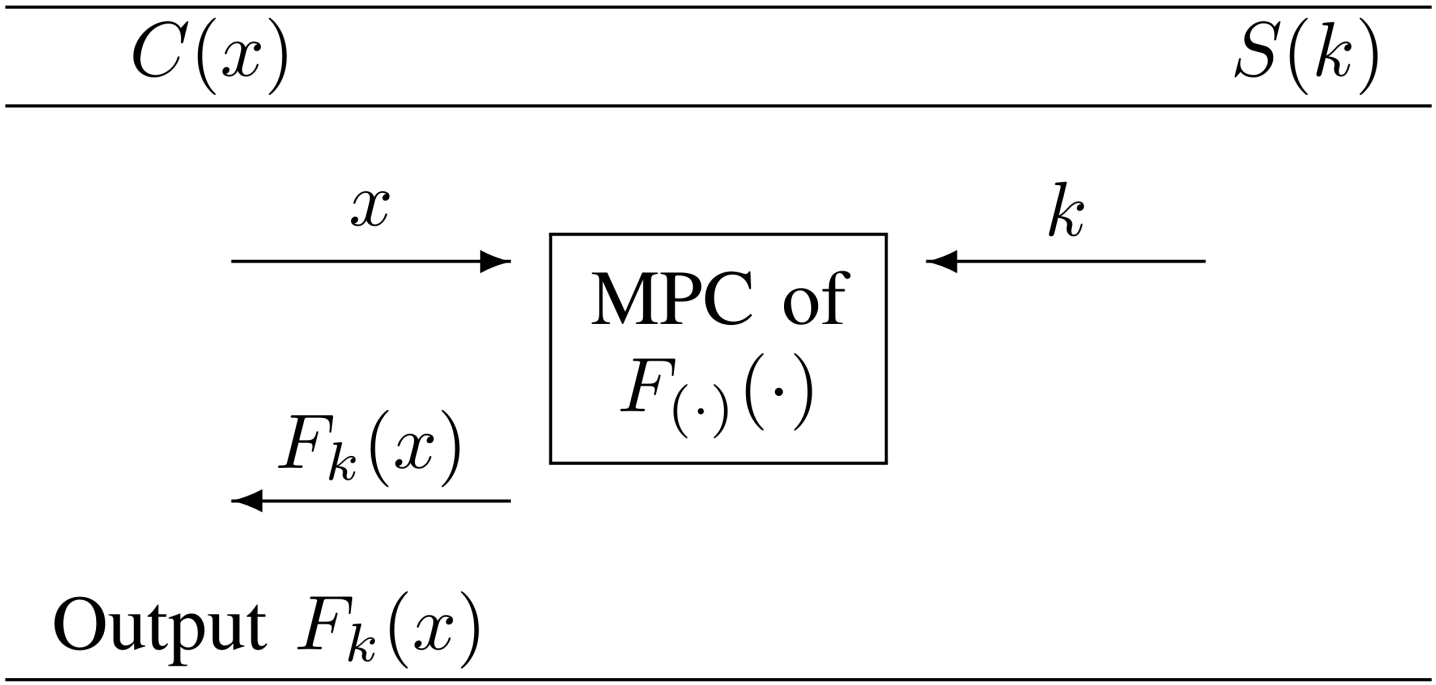
Dodis-Yampolskiy PRF

$$f_k^{DY}(m) := g^{\frac{1}{k+m}}$$

Dodis-Yampolskiy OPRF

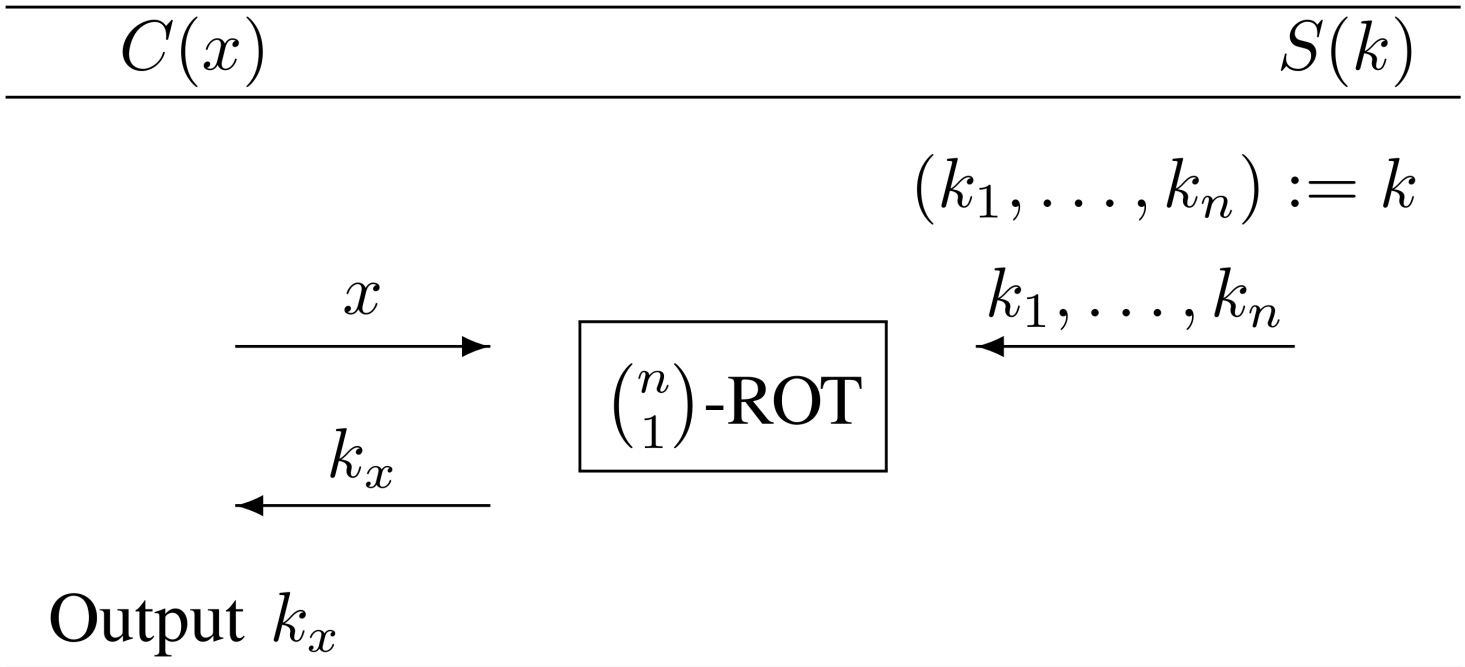
$C(x, \mathsf{pk})$		$S(k, \mathsf{sk}, \mathsf{pk})$
	$\mathsf{HEnc}_{\mathsf{pk}}(k)$ $\longleftarrow$	
$r \leftarrow_{\mathrm{R}} \mathbb{Z}_q$	$(\mathsf{HEnc}_{\mathsf{pk}}(k) \cdot \mathsf{HEnc}_{\mathsf{pk}}(x))^r$ $\longrightarrow$	Decrypt to $r(k+x)$ $y' \leftarrow g^{1/r(k+x)}$
Output y'^r	y' $\longleftarrow$	

Generic MPC OPRF



f = AES - is also PQ!

Generic OT OPRF



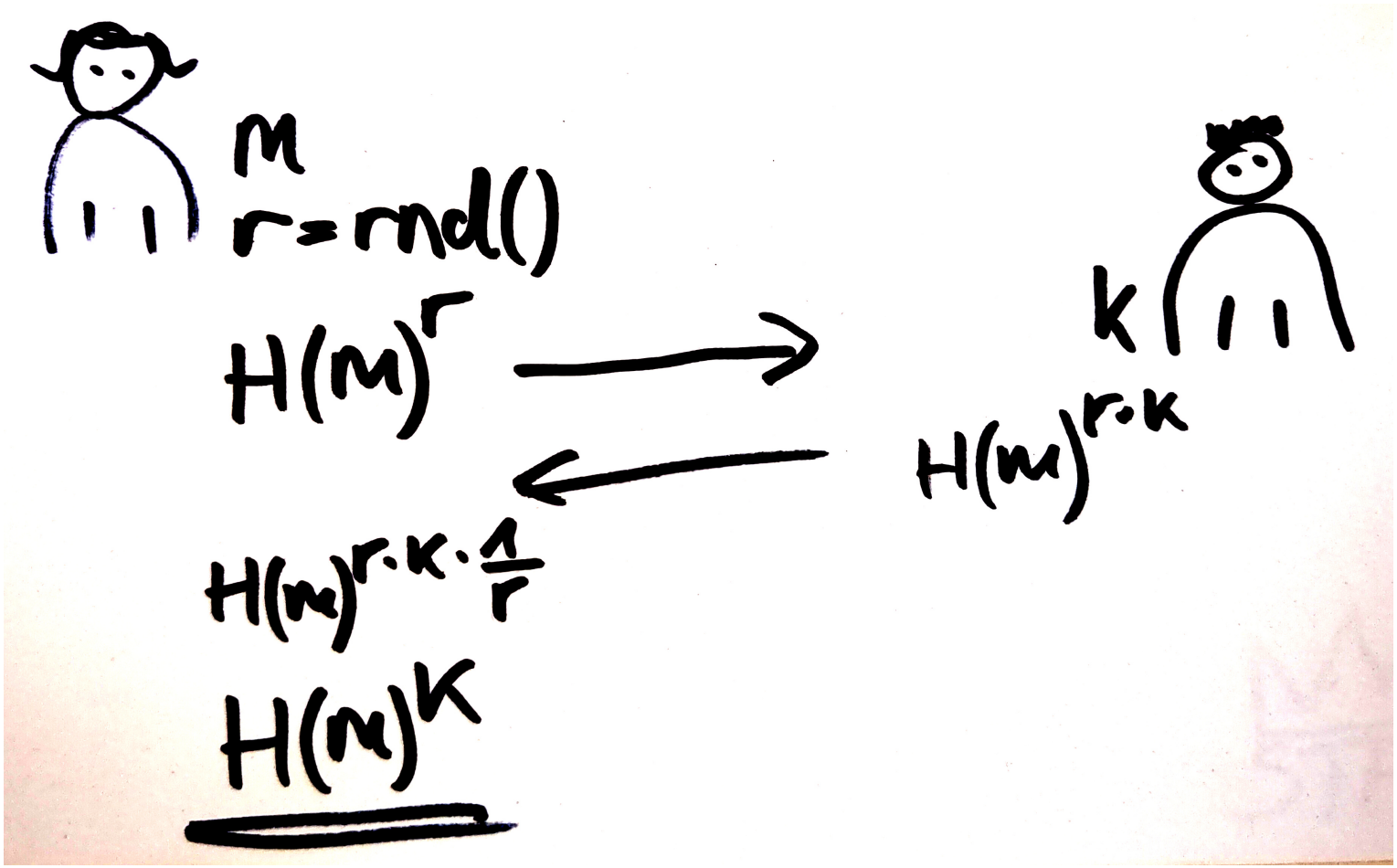
HashDH

$$f_k^H(m) := H(m)^k$$

2HashDH

$$f_k^{2H}(m) := H'(m, H(m)^k)$$

Protocol



Properties

	PRF	Batching	Multi-point	Partially oblivious	Verifiable	Committed	Proactive sec.	Updatable	Distributed	Threshold	Special (Sect. 3.8)
FIPR05, 1st [4]	NR	○	●	○	○	○	○	○	○	○	○
HL08 [5]	NR	○	●	○	●	○	○	○	○	○	○
JKK14 [9]	NR	○	●	○	●	○	○	○	○	○	○
Hazay15 [26]	NR variant	○	●†	○	●	○	○	○	○	○	○
JL09 [7]	DY	○	●	○	●	●^{i}	○	○	○	○	○
CL17 [22]	DY	○	●†	○	○	●^{o}	○	○	○	○	○
MPRSY20 [30]	DY	●	●	○	●	○	○	○	●	○	○
TCRSTW21 [31]	DY variant	○	●	●	●	○	○	○	○	○	○
JL10 [8]	2HashDH	●†	●	○	○	○	○	○	○	○	○
JKK14 [9]	2HashDH/RSA	○	●	○	●	○	○	○	○	○	○
ECSJR15 [19]	HashDH	○	●	●	●	○	○	●	◌	●	○
JKKX16 [10]	2HashDH	○	●	○	○	○	○	○	○	○	○
JKKX17 [11]	2HashDH	○	●	○	○	○	○	○	◌	●	○
DGSTV18 [28]	2HashDH	●†	●†	○	●	○	○	○	○	○	○
Lehmann19 [57]	HashDH	●♣	●	◌	○	○	○	◌	○	○	●
BFHLY19 [12]	2HashDH	○	●	●	○	○	●	○	●	○	○
DHL22 [13]	2HashDH	○	●	●	○	○	○	○	●	○	●
KKRT16 [6]	Any	●✶	○	○	○	○	○	○	○	○	○
KMPRT17 [34]	Any	○	○	○	○	○	○	○	○	○	●
JKR18 [58]	Any	○	●◇	●	●	○	○	●	◌	●	○

TABLE 3: Properties of OPRFs from the literature.

● = applicable
◌ = trivially implied
○ = not satisfied

✶ = with related keys
† = enforcing same key
◇ = input-dependent key

♣ = for same input
sh = semi-honest server
i = input, o = output

src: SoK OPRFs

Partially-oblivious PRFs (POPRF)

$$f_k(m_{priv}, m_{pub})$$

- initially bilinear parings-based
- $f_k^{modDY}(m_{priv}, m_{pub}) := H`(m_{priv}, m_{pub}, H(m_{priv})^{\frac{1}{k+m_{pub}}})$
- $f_k(m_{priv}, m_{pub}) := f'_{PRF(k,m_{pub})}(m_{priv})$

Verifiable OPRF (VOPRF)

- Client can verify that the server used k correctly, and that it is the same over repeated OPRF invocations.
- Either by publishing a "public key" of k, or some zero-knowledge proof.

Committed I/O

- Client commits to the input value or the output value using Pedersen commitments.
- This can prove that the input to the OPRF is a value from an previously generated value in the Protocol.
- For outputs the commitment can prove in later steps of a protocol, that the value is the output of the OPRF.

Updateable OPRF

Updateable OPRFs allow for key rotation, in order to minimize the risk and impact of key exposure. It allows a client to update their result of a previous OPRF run with a new key of the server.

1. assuming the OPRF calculates m^k
2. server updates k to k', calculates $\Delta = \frac{k'}{k}$ and sends Δ to the client
3. the client updates a previous calculation of $f_k(m)$ by raising it to Δ: $f_{k'}(m) = f_k(m)^{\Delta} = m^{k\,\frac{k'}{k}}$

This can provide proactive or post-compromise security.

Distributed and Threshold OPRFs (t-OPRF)

- additive sharing $\rightarrow$ Distributed OPRFs
- DH-based OPRFs + Shamir's secret sharing $\rightarrow$ t-OPRFs

provides resilience against:

- denial of service attacks or loss of SPoF
- key compromise
- protection for low entropy inputs

Batching

Multiple parallel execution of an OPRF using either the same key or input, while being cheaper than running all of these sequentially. This is most useful for for private set intersection.

Other properties

- Weak OPRFs
- Programmable
- 3-party
- Convertability
- Extendable

Hey, what about pqOPRF?

well.

- "HashDH-based OPRFs information-theoretically blind the input of the user, and hence protect it even against quantum computers." - but not the key...
- Round-optimal Verifiable Oblivious Pseudorandom Functions From Ideal Lattices `https://eprint.iacr.org/2019/1271`
- Oblivious Pseudorandom Functions from Isogenies

  `https://eprint.iacr.org/2020/1532`
- OPRFs from Isogenies: Designs and Analysis

  `https://eprint.iacr.org/2023/639`
- Crypto Dark Matter on the Torus: Oblivious PRFs from shallow PRFs and FHE `https://eprint.iacr.org/2023/232`
- A Post-Quantum Round-Optimal Oblivious PRF from Isogenies `https://eprint.iacr.org/2023/225`
- OT OPRF with AES is PQ! but does not support many properties

Which to use?

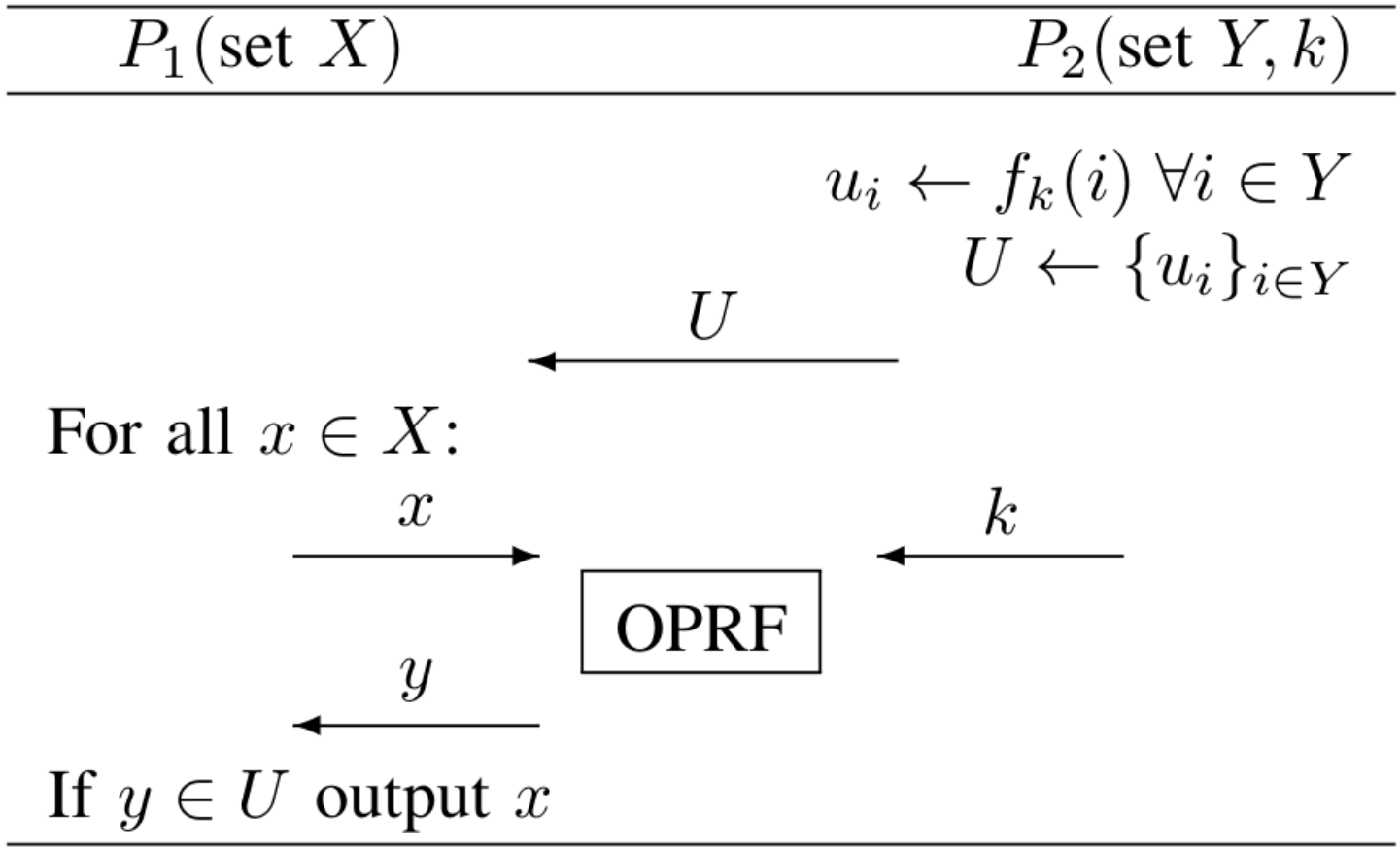
- DY if you need I/O committed, or need partial-oblivious verifiability
- OT+AES if you need speed and no special properties
- 2HashDH for everything else, unless you need updateability, then HashDH.

POPRF, VOPRF and OPRF IRTF/CFRG specification

- `https://github.com/cfrg/draft-irtf-cfrg-voprf/`
- `https://datatracker.ietf.org/doc/draft-irtf-cfrg-voprf/`

HashDH based, depends on hash-to-curve - which is difficult.

PSI



- G Password Checkup - similar to haveibeenpwned
- Contact Discovery (Wickr [10])

[10] https://wickr.com/better-contact-discovery/

SSO

- "PESTO: Proactively Secure Distributed Single Sign-On, or How to Trust a Hacked Server[11]" uses a dpOPRF with bilinear pairings for the partial obliviousness and a DSIG, compatible with OATH and OIDC.

[11] https://eprint.iacr.org/2019/1470

SPHINX[12]

- Information Theoretically secure password "store"
- "Could be hosted by TLA" - and your passwords would still be safe!
- Server does know nothing about input or output password!
- Resistant to offline bruteforce attacks!
- No encryption! Leak, much? No problem!!5!
- No encryption! No global encryption key! Can use multiple master passwords!
- KISS! does one thing, but does it right!
- `https://ctrlc.hu/~stef/blog/posts/sphinx.html`
- `https://github.com/stef/pwdsphinx/`

in debian and derivatives!

[12] `https://eprint.iacr.org/2018/695`

SPHINX

1. if you don't need multi-device support just do rwd=PRF(key, concat(password,user,host))
2. if you need multi-device support then using 2HashDH is SPHINX: rwd=OPRF(k,password)
3. You can convert the binary output to ASCII strings conforming to password rules using simple conversions.
4. Using a random pad, you can even force short desired outputs from the binary to ASCII conversion, by having "$desiredoutput$" $= rwd \oplus pad$

OPAQUE[19]

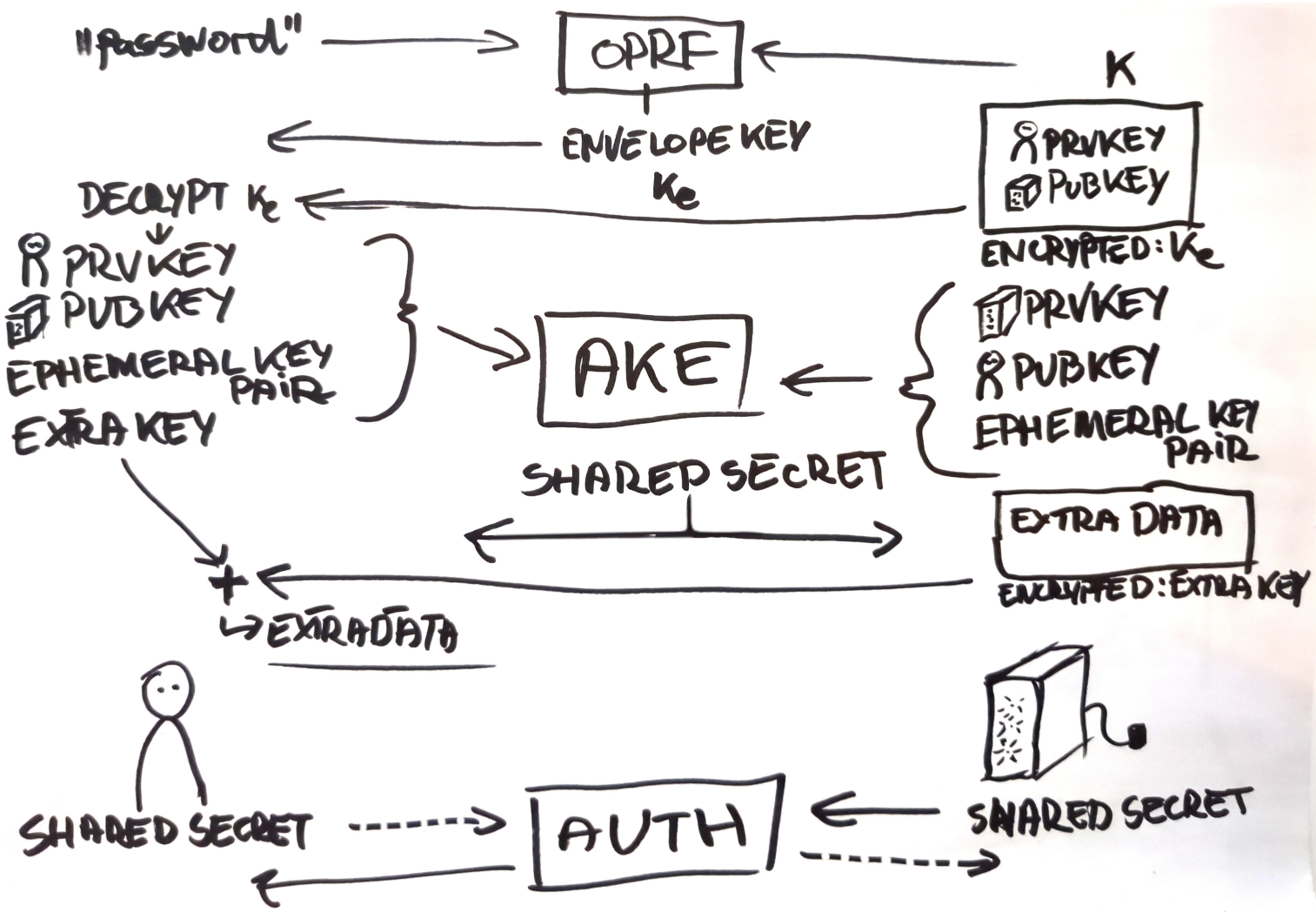
OPRF + AKE used by

- IRTF/CFRG draft[13, 14]
- libopaque[15]
- opaque-store[16] - 1st public threshold-opaque implementation!!5!
- TLS-OPAQUE[17]
- WhatsApp backup[18]

---

[13] https://github.com/cfrg/draft-irtf-cfrg-opaque/
[14] https://datatracker.ietf.org/doc/draft-irtf-cfrg-opaque/
[15] https://github.com/stef/libopaque/
[16] https://github.com/stef/opaque-store/
[17] https://eprint.iacr.org/2023/220
[18] https://www.whatsapp.com/security/WhatsApp_Security_Encrypted_Backups_Whitepaper.pdf
[19] https://eprint.iacr.org/2018/163

OPAQUE


"password"
OPRF
K
ENVELOPE KEY
K_e
DECRYPT K_e
PRVKEY
PUBKEY
ENCRYPTED: K_e
PRVKEY
PUBKEY
EPHEMERAL KEY PAIR
EXTRA KEY
AKE
PRVKEY
PUBKEY
EPHEMERAL KEY PAIR
SHARED SECRET
EXTRA DATA
ENCRYPTED: EXTRA KEY
+
↳ EXTRADATA
SHARED SECRET
AUTH
SHARED SECRET

VTUOKMS[21] aka Klutshnik

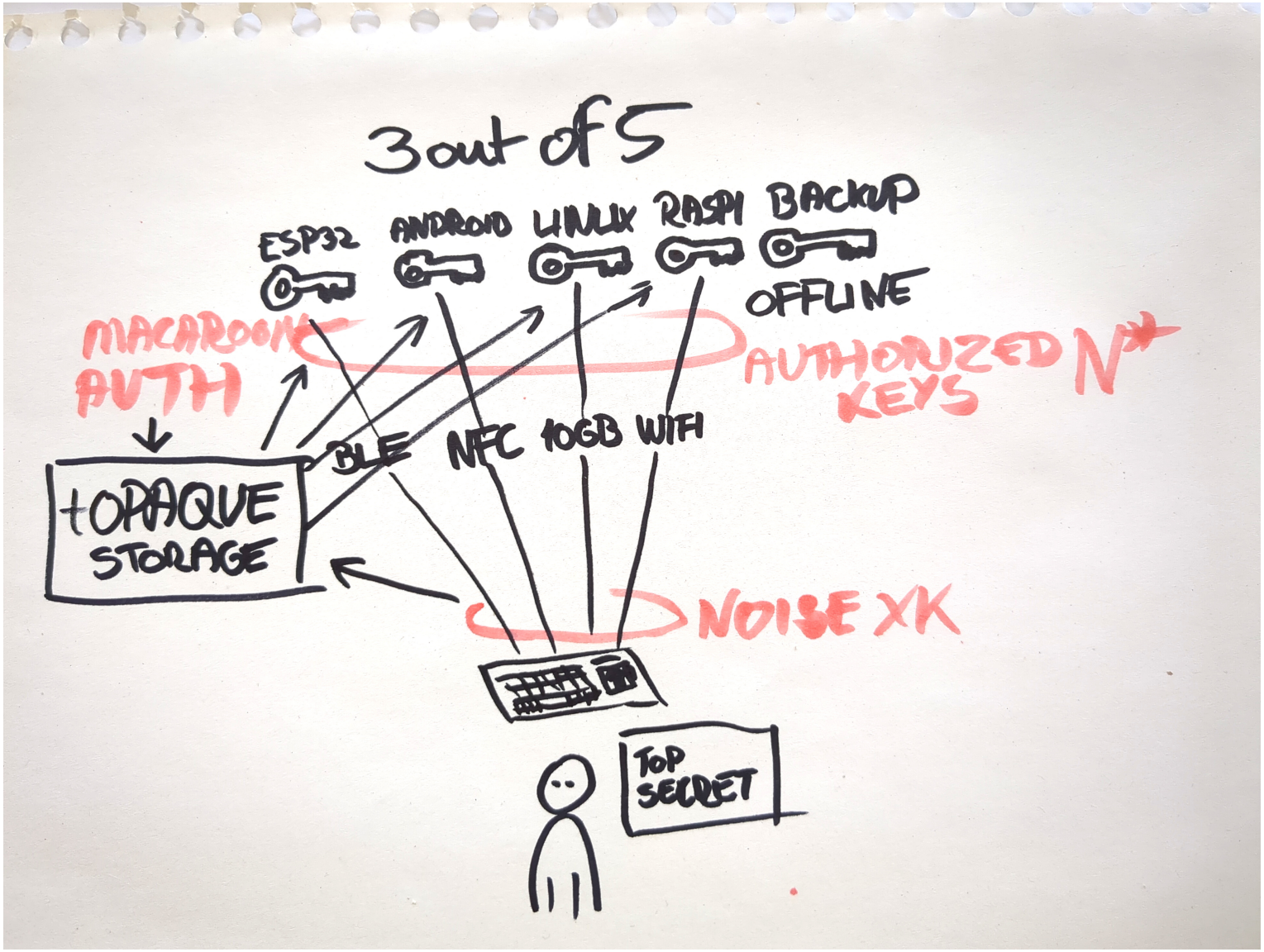
strong authentication. `https://klutshnik.info/`

- similar: "DPaSE: Distributed Password-Authenticated Symmetric Encryption"[20] uses an extendable dpOPRF with bilinear pairings for the partial obliviousness. Cannot be threshold, and the second evaluation is not verifiable.

[20] `https://eprint.iacr.org/2020/1443`

[21] `https://eprint.iacr.org/2019/1275`

Klutshnik


3 out of 5
ESP32
ANDROID
LINUX
RASPI
BACKUP
OFFLINE
MACAROONS
AUTH
AUTHORIZED KEYS
N
+OPAQUE STORAGE
BLE
NFC
10GB WIFI
NOISE XK
TOP SECRET

Privacy Pass[23]

blind signature, to see less captchas when using tor

1. server publishes "long-term" g^k
2. user mints: t^k, zkp(k) = VOPRF(t,k)
3. user redeems: $t, M, HMAC(t^k, M) \rightarrow$ *server*
4. server allows if $HMAC(t^k, M) == HMAC(PRF(t, k), M)$ and notes down t to prohibit double-spending

But note:

> *Landau's Law: A cryptosystem is incoherent if its implementation is distributed by the same entity which it purports to secure against.*[22]

[22] https://devever.net/~hl/webcrypto

[23] https://privacypass.github.io/protocol/

Summary

OPRFs are cool!!5! Use/demand them:

- When you care about privacy!
- When you have 2 parties.
- Use when you want a lightweight client - no or minimal data and/or computation.
- When you want to protect the data on the client from the server.
- When you want to convert a low-entropy input to a high-entropy strong cryptographic value.
- When you need Interactive hashing - e.g. for rate-limiting, prohibit pre-computation.

?

Questions? Comments!

TEST DELETE BEFORE PUBLISHING