Threshold Cryptography: Ready for Prime Time?

Hugo Krawczyk, IBM

NIST Threshold Cryptography Workshop 2019

3-11-2019

(with thanks to many colleagues and collaborators – see references at the end)

Threshold Cryptography: Ready for Prime Time?

- I would say YES! (except that I said the same 20 years ago…)

  - Huge increase in quantity and sensitivity of stored data
  - Cloud storage and computing as the prevalent paradigm
  - Huge key management operations (see Campagna’s AWS talk at RWC’19)
  - Privacy concerns, awareness, regulations
  - Awareness of dangers of centralization: Facebook, Google, Amazon, *WeChat*
  - *Distributed trust* becoming a more “familiar” notion via blockchain
  - Advances in cryptography: MPC, homomorphic techniques, ZK, …
  - **New applications!** THIS WORKSHOP

With great opportunities come great challenges

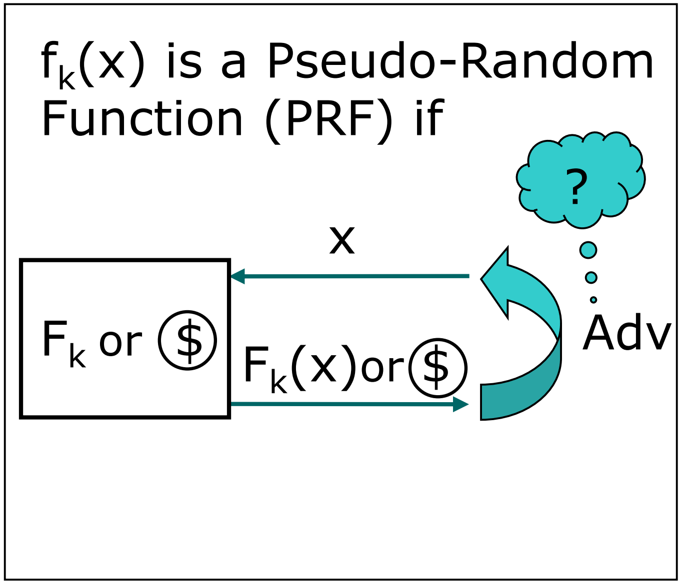
- The distributed trust notion (a hard one to reason about)
- True distribution → *diversity*: O/S, h/w, geography, tooling, admin, policy and authorization, credential management,… → fault independence
- Wide Area Networks, asynchronous networks, *going beyond n/3* !
- Role of secure h/w (enclaves, HSMs), virtualization, side channel sec.
- Distributed key generation, share recovery, proactive, *identifying cheaters*
- Integration with MPC, blockchains, ZK proofs, …
- Large scale TC (10's/100's/1000's/millions parties?)
- Post quantum techniques, including symmetric crypto and inf. Theoretic (NIST competition: Prioritize schemes with threshold implementations)

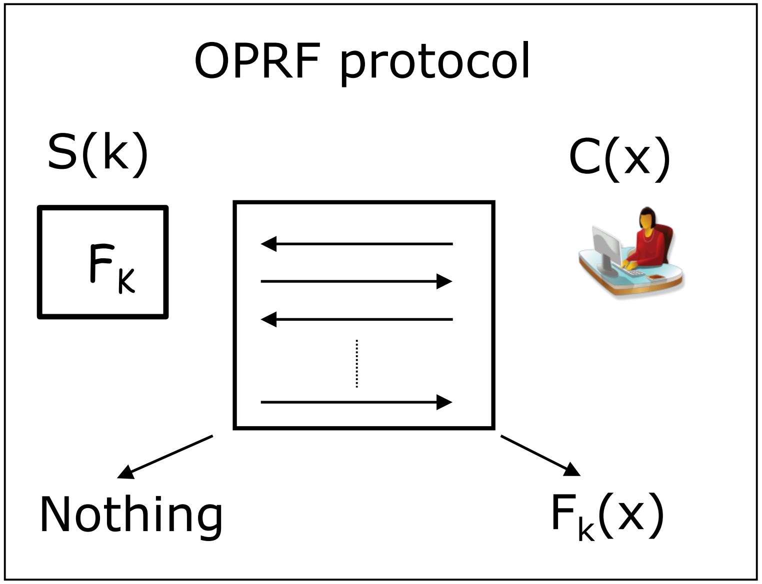
Your favorite challenge here

Mine: Build a serious open source platform for threshold cryptography

Demonstrate new *practical* applications!

Oblivious PRF (OPRF)





- OPRF: An interactive PRF "service" that returns PRF results *without learning the input or output of the function*
- *A POWERFUL primitive*

DH-OPRF

$H'(x, H(x)^k)$

[CP93….NPR99…FIPR05 …JL10…JKK14…]

- **PRF**: $F_K(x) = H(x)^k$; input x, key k in Z_q ; H = RO onto G (of order q)
- Oblivious computation via Blind DH Computation (S has k , C has x)

S: key k | **C:** input x

random r

$a = (H(x))^r$ ←-------------------

$b = a^k$ -------------------→

Computes $H(x)^k \leftarrow b^{1/r}$

- The blinding factor r works as a one-time encryption key: ***hides*** $H(x)$, x ***and*** $F_K(x)$ ***perfectly from S*** (and from any observer)
- Computational cost: one round, 2 exponentiations for C, one for S
  - Variant: fixed base exponentiation for C (even faster)

Threshold DH-OPRF

- Single server solution: $F_k(x) = (H(x))^k$
- Multi-server solution: server S_i initialized with (t,n)-share k_i
- Shamir in the exponent (polynomial interpolation)
  - $F_k(x) = (H(x))^{\lambda_{i1}k_{i1}} \cdot (H(x))^{\lambda_{i2}k_{i2}} \cdot \ldots \cdot (H(x))^{\lambda_{i,t+1}k_{i,t+1}}$
  - C sends same $a = (H(x))^r$ to $t+1$ servers;
  - S_{ij} raises $a^{\lambda_{ij}k_{ij}}$ and sends back to U who deblinds and multiplies *
- Efficiency (!): 2 exp's for client (indep of t, n), 1 per server, 1 round

* If responders among servers not known a-priori, interpolation done by U

. (one multi-exponentiation; can be further optimized [Patel-Yung])

Threshold DH-OPRF (more features)

- Threshold operation *transparent to client*
  - Client sends one and same msg to all servers and aggregation of $a^{k_{ij}}$ to a^k can be done by a single server (proxy)
- Distributed key generation (key never exists in one physical place)
- Share recovery, Proactive security (fundamental for long-lived keys)
- Verifiability: With g^k, C can verify that $H(x)^k$ computed correctly
  - Preserves client transparency using interactive verification (2x cost)
  - Can also use BLS for "built-in verifiability"

Proving Threshold DH-OPRF [JKKX'17]

- UC Definition of Threshold OPRF: Extends the (single) OPRF UC formulation of JKKX'16
- Ticketing mechanism: increases when threshold of servers responds; decreases when client reconstructs an output
  - Avoids extraction and other proof elements that degrade performance
- Proof of Threshold DH-OPRF based on Gap-OMDH assumption in ROM, and on Gap-TOMDH to achieve a stronger flavor
  - OMDH: "Q interactions with $(\cdot)^k$ ➔ No more than $g_1^k,\ldots,g_Q^k$ on random g_i "
  - T-OMDH: require t+1 online attempts for each g_i^k .

PPSS: Password Protected Secret Sharing

(password-protected distributed storage)

How to protect a secret with a password

- Goal: protect *secrecy* and *availability* with a single password
  - Single server = Single point of compromise for secrecy (offline dict attacks) and for availability (server gone, secret gone) ➔ multi-server solution
- Crypto solution: keep the secret encrypted in multiple locations; secret share the encryption key in multiple servers (t-out-of-n)
  - Availability insured if t+1 available, secrecy if t or less corrupted
- But how do you authenticate to each server for share retrieval?
  - A strong independent password with each server? Not realistic
  - Same (or slight-variant) password for each server? Not good

➔ Each server is a single point of compromise!

How to protect a secret with a password

- Password-Protected Secret Sharing (PPSS) guarantees
  - Breaking into t servers leaks nothing about secret or password (assumes all server info lost: shares, long-term keys, password file, etc.)
  - Only adversary option: Guess the password, try it in an online attack.
- Definition [BJSL'12, CLLN'14, JKKX'16]
  - Only *unavoidable online* attacks allowed: Attacker needs at least t+1 *online* interactions to validate a single guessed password
  - Offline attacks are not possible, except if t+1 servers compromised
  - Subtlety: User needs a way to verify the reconstructed secret is correct (w/o that information allowing offline attacks)

Important: No PKI

TOPPSS: PPSS via Threshold OPRF [JKKX'17]

- Idea: Define the retrieved secret as $s = OPRF_k(pwd)$
  and implement the computation as a Threshold OPRF
  - U: send $a=H(pwd)^r$, get a^{Ki}, reconstruct $s= H(pwd)^K$ + mechanism to test s
- Definitions and analysis tricky but protocol very simple
  - Crucial detail: Must be able to verify the correct secret reconstruction
  - Note: No PKI reliance (except for initialization)
- PPSS performance: same as Threshold DH-OPRF
  - Single round, total 2 exp for client, 1 exp for each server, client transparent
- Proactive security and other goodies (as in underlying T-OPRF)

From (t,n)-PPSS to (t,n)-threshold PAKE

- (t,n)-TPAKE [MSJ'02]: Single-password PAKE b/w U and any subset of n servers - *secure* as long as at most t servers are corrupted
  - Addresses the main threat to passwords today, namely, leakage via server compromise (even t adversarial servers learn nothing about password)
- *Generic composition theorem:* PPSS + KE → T-PAKE [JKK14]

→ *First single-round T-PAKE and best computational performance (2 exp user, 1 exp server)*

  - Best previous work required 10 msgs plus 14t exponentiations for client and 7t for each server (even a dedicated 2-out-of-2 sol'n required 5 msgs)

More Password Applications from (T-)OPRF

- OPAQUE = "an asymmetric (1,1)-PAKE" (hopefully integration w/TLS 1.3)
  - Much more secure than "password-over-TLS" (pwd never exposed)
    - ***First*** client-server PKI-free PAKE secure against pre-computation attacks!
  - Server can be implemented as **threshold OPRF**: Best protection against server compromise and offline attacks (the way most passwords are stolen)
- SPHINX: Server-based online password manager
  - User only remembers master password, interacts with SPHINX server(s) to create *random independent* passwords for each of its accounts
  - Magic property: Breaking into the server leaks *nothing* on the user's master password or on the random account passwords
    - after breaking the server an online guessing attack is still required

OPRF-based Key Management

- Ciphertexts and keys need to be stored separately. How? Client stores ciphertexts, outsources the key to a key management server (KMS)
- Today: All encryption keys exposed to the KMS and to channel between KMS and client (e.g., tls failures, certificates, termination points, CDN,…)
- Using OPRF: KMS learns ***nothing*** about key or object being encrypted, and neither do observers of client-KMS channel (unconditional security)
- Plus: If client assigns unpredictable identifiers to objects
  → forward security (keys remain secure upon full compromise of KMS)
- It gets better: **Threshold and proactive security!!**

(Oh. And non-interactive key rotation.)

Threshold Decryption

- General use case: Data encrypted under a service public key; decryption possible only upon collaboration of t servers
  - Data protected up to the compromise of t servers
- Examples
  - Long-term data storage: sensitive and valuable information, e.g. personal information, legal and financial documents, cryptographic keys, etc.
  - Computation on encrypted data, only decrypt results (e.g., voting, FHE)
  - Specialized cases of computation on encrypted data
    - Next: Two such examples

Operations on Encrypted Sets

- Set representation using polynomials [FNP'04, KS'05]
  - Set of elements a_1, …, a_n represented by n-degree polynomial $(x-a_1) \cdots (x-a_n)$
  - Membership test: a is in the set iff P(a)=0
  - Adding an element: If P(x) represents S, P'(x)=P(x)(x-a) represents S' = S ∪ {a}
- Privacy preserving operations: Encode coefficients using linear homomorphic encoding (via Elgamal encryption)

- Define Elgamal encoding $E(v) = EG_h(g^v) = (g^k, h^k g^v)$ under PK h.
- $P(x) = \sum_{i=0}^{n} P_i x^i$ represented as encoding of coefficients

$$E(P_i) = EG(g^{P_i}) = (g^{k_i}, h^{k_i} \cdot g^{P_i})$$

Operations on Encrypted Sets

- Element addition: $P'(x) = P(x)(x-a)$

$$P'_i = P_i - aP_{i-1};\quad E(P'_i) = EG(g^{P_i}) \cdot (EG(g^{P_{i-1}}))^{-a}$$

- Membership test: $a \in S$ iff $P(a) = 0$

$$E(P(a)) = E(\sum_{i=0}^{n} P_i a^i) = EG(g^{\sum_{i=0}^{n} P_i a^i}) = \prod_{i=0}^{n}(EG(g^{P_i}))^{a^i}$$

  - Given $E(P_0), \ldots, E(P_n)$ and a, compute $C = \prod_{i=0}^{n}(EG(g^{P_i}))^{a^i}$
  - **Decrypt C and conclude $a \in S$ iff result is 1.**

- In our applications, test uses **threshold decryption**

  Note: nothing learned about the values of P_i, only whether $P(a) = 0$

  (also note that coefficient decryption is not possible)

Digital Asset Transfer in Blockchain

- Example: Know Your Customer (KYC)
- Bank A performs KYC for customer U while opening account
  - U and A *own* the KYC file; A is willing to share it with other parties, e.g. bank B, upon *U's request* and upon *payment* by the receiving party
  - U does not want A and B to know each other's identity (for *privacy*)
  - U wants to remain *anonymous* to any party other than A and B and wants repeated uses of KYC to remain *unlinkable*.
  - A does not want another entity (e.g., bank B) to sell U's KYC – doing so represents *counterfeit* by Bank B (even if done in collaboration with U)
- Blockchain solution helps to enforce all the above properties (pseudonyms, commitments, payment, recording, ZK proofs, …)

Counterfeit/Duplicate Prevention

- Set of submitted values a (asset hashes) is recorded in blockchain via an encoded polynomial (i.e., encoded coefficients committed to b/c)
- Submitter of asset hash a, computes ciphertext C (encoding of P(a)) and an encoding of P(x)(x-a) (using public homomorphic operations)
  - Submitter proves in ZK correct computation with respect to a committed (and hidden) value a
- Blockchain peers **threshold decrypt** C (after randomizing it)
  - If result is 1, they reject value a (as already recorded)
  - Otherwise, they update the encoded-coefficients in blockchain to those submitted for P(x)(x-a)

Cryptography for #MeToo

- Most sexual assault is perpetrated by repeat offenders
- Goal: Identify survivors of same perpetrator while protecting anonymity of accusers and accused *except if #accusations > quorum*
- Ideal functionality: Accusers submit a (accuser-id, perpetrator-id) accusation to a trusted third party who matches perpetrators, and contact survivors if count for an accused goes over the quorum.
- We show a solution that achieves such functionality with full privacy

Solution via Threshold Decryption

- Use encoded/encrypted polynomials to **encode a multi-set**
- Accusation against accused A recorded as encoding of P(x)(x-a)
- Accused A reaches Q accusations when $(x-A)^Q/P(x)$
- Testing $(x-A)^Q/P(x)$ via randomized (Q-1)-th derivative of P(x)
- Reduces to checking $P^{(Q-1)}(A) = 0$ (derivative is "homomorphic")
  - implemented via public operations on encoded polynomials and a single membership test using decryption (as in BC example but more involved)
- Test performed via **threshold decryption** by a set of dedicated servers

Concluding Remarks

- We live in exciting times
- The world cries for distributed cryptography (even if they don't know it)
- Threshold cryptography is one of the most useful and practical branches of MPC - great applications!
- Varied, interesting, timely, practical, ready-to-deploy solutions
- Many challenges ahead, a lot to invent…
- … and to implement and deploy (open source, please)
- Important role for NIST: Credibility, visibility, motivation
  - Standards and **best practices** as inputs to regulations

Works Mentioned and Colleagues

- TOPPSS: S. Jarecki, A. Kiayas, H. Krawczyk and J. Xu, eprint.iacr.org/2017/363
- T-PAKE: S. Jarecki, A. Kiayas, and H. Krawczyk, eprint.iacr.org/2014/650
- OPAQUE: S. Jarecki, H. Krawczyk and J. Xu, eprint.iacr.org/2018/163
- SPHINX: M. Shirvanian, S. Jarecki, H. Krawczyk, N. Saxena, eprint/2018/695
- KMS: S. Jarecki, H. Krawczyk and J. Resch, eprint.iacr.org/2018/733 (preliminary)
- Asset transfer: H. Gunasinghe, A. Kundu, E. Bertino, H. Krawczyk, K. Singh, S. Chari, D. Song, The Web Conference, WWW'2019.
- MeToo: B. Kuykendall, H. Krawczyk, and T. Rabin, PETS 2019.
- Beyond n/3: C. Cachin, H. Krawczyk, T. Rabin, J. Resch, C. Stathakopoulou, "Tunable Protocols for Threshold and Proactive Cryptography", coming soon.
- **A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, F. Valsorda: Privacy Pass: Bypassing Internet Challenges Anonymously, PETS'18**

Only example of deployed OPRF?

Thanks!

Define "Threshold Cryptography"

- Threshold Cryptography: A special case of secure multi-party computation (MPC).
- TC characterized by the *distribution of a centralized service* for protecting both secrecy and availability.
  Clients can think of the service as one unit.
  - TC as an "implementation issue", behind the scenes (the less the client is aware of it, the better – client transparency – communication via gateway)
  - The MPC happens at the servers, clients do not run an MPC or talk among themselves (though they may talk to a set of servers – e.g., for decryption)
- TC as one of the most useful and easier to motivate MPC flavors