The OPAQUE Password Protocol

Authentication, Secret Retrieval, End-to-end security

Hugo Krawczyk, AWS

NIST Seminar – 10.16.2024

Joint work with: Stas Jarecki, Jiayu Xu (paper),
Daniel Bourdrez, Kevin Lewi, Chris Wood (RFC)

Password (In)Security

- Passwords: Dominant authentication tool in the digital era
- Protect our data, lives and social order, *conveniently* and **Insecurely**
- BILLIONS of passwords stolen
- Millions logged *in the clear* accidentally
  - Facebook, Google, Twitter, Github, …


GOOGLE \ TECH \ CYBERSECURITY

Google stored some passwords in plain text for fourteen years

Only affects some G Suite customers

By Dieter Bohn | @backlon | May 21, 2019, 7:16pm EDT

Attacks

Unavoidable attacks

- *Online* guessing: mitigation via throttling, 2nd factor
- *Offline* dictionary attack *upon server compromise* ("password file")
  - E.g., given (salt, Hash(salt,pwd)) check Hash(salt,guess)
  - **All** pwd protocols are open to offline attack by the server (simulate user)

Avoidable attacks (e.g., password-over-TLS)

- Password visible to server
- PKI dependency: Password compromised with PKI failure
- Phishing: User sending password to the wrong server

Designing a Secure Password-Authenticated Key-Exchange Protocol

What We Want

- *Client-Server setting*: Server has no access to the password, only stores a one-way image of user's password
- No pre-computation attacks (unavoidable brute force dictionary attack *but attack starts only upon server compromise*)
- Plaintext password *never visible to server (incl. at registration)*
- *PKI free*: user remembers password, nothing else (*excl. registration*)
- Not only authentication but *secure channel establishment* (key exch.)

aPAKE: Asymmetric Password Authenticated Key Exchange

(asymmetric = client-server; a.k.a. 'augmented')

What We Want

- *Client-Server setting*: Server has no access to the password, only stores a one-way image of user's password
- No pre-computation attacks (unavoidable brute force dictionary attack *but attack starts only upon server com*
- Plaintext password *never* *ver (incl. at registration)*
- *PKI f* embers password, nothing else (*excl. registration*)
- Not only authentication but *secure channel establishment* (key exch.)

Essentially, nothing but unavoidable attacks

aPAKE: Asymmetric Password Authenticated Key Exchange

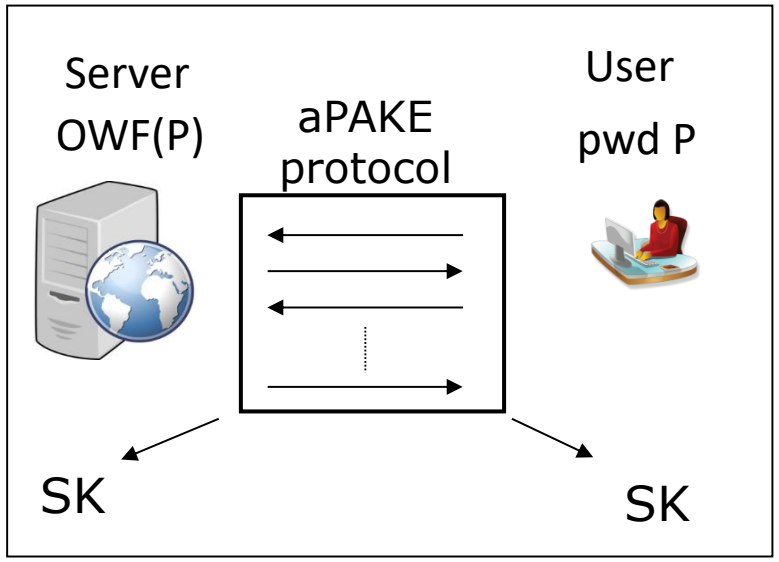
(asymmetric = client-server; a.k.a. 'augmented')

Why Key Exchange?

Asymmetric Password Authentication Key Exchange (aPAKE)

(Later, also password-based key/credential retrieval)

Asymmetric Password-Authenticated Key Exchange (aPAKE)



- All or nothing property:
  - The attacker knows password: Impersonation unavoidable
  - Attacker does not possess the password:

As secure as full-entropy KE!

- Only unavoidable attacks are allowed:
  - Online guessing: attempt to impersonate user with guessed passwords (one guess verification per interaction)
  - Compromise the server's state and run an exhaustive offline dictionary attack

Password-over-TLS is NOT a Secure aPAKE!

- Password over TLS
  - Client establishes server-authenticated secure channel with server (using server certificate)
  - Password sent from client to server encrypted by this channel
  - Server decrypts password and checks it against a stored pair (salt,hash(salt,password))
- Two main weaknesses
  - Assumes user/client has a way to verify certificates: We want **password-only** protocols.
  - Plaintext password is visible to the server (and to any decrypting mid/end-point)

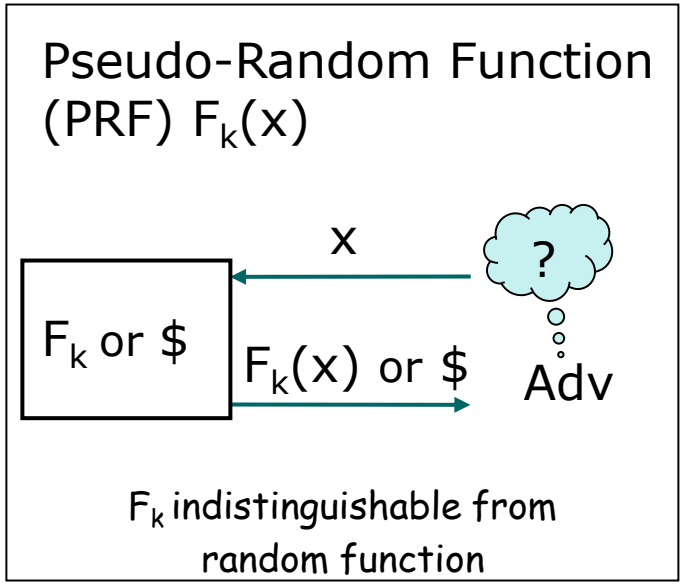
aPAKE: PKI-free authentication and password never visible to server

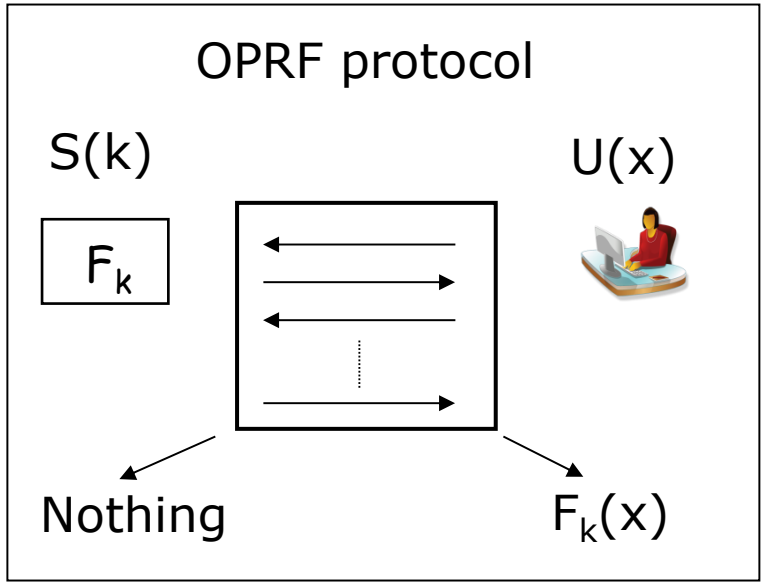
We will build a secure aPAKE protocol

But first…

Oblivious PRF

Oblivious PRF (OPRF)





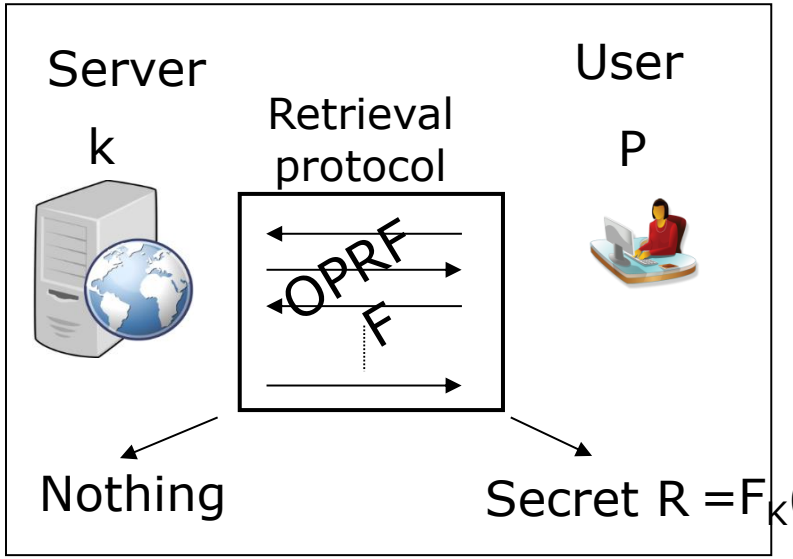
User learns $F_k(x)$ and *nothing else* and server learns *nothing* (neither x or $F_k(x)$)

Oblivious PRF (OPRF)

- Beautiful and powerful cryptographic primitive with numerous applications
  - Private set intersection
  - Keyword search and searchable encryption
  - Private access tokens (e.g., PrivacyPass)
  - Forward secure key management system
  - […]
- **Here we use them to build password protocols**
- **Later we show VERY simple and efficient OPRF implementation**

OPAQUE

OPAQUE: Trading a Password for a Strong Secret via OPRF



OPAQUE aPAKE protocol:

- U runs OPRF on input password P with server S to retrieve random secret R
- U uses R as a private key in a regular key exchange protocol with S

OPAQUE with key exchange protocol KE and OPRF $F_K(\cdot)$

- KE: PK-based authenticated key exchange; uses $(priv_S, pub_S)$ and $(priv_U, pub_U)$
- Registration (of user U at server S):
  - S chooses fresh OPRF key k and a pair $(priv_S, pub_S)$ for protocol KE; sends pub_S to U.
  - U runs OPRF with S on input password P to learn secret $R=F_k(P)$;

    U derives KE keys $(priv_U, pub_U)$ from R and sets $Env_U = MAC_R(pub_S)$
  - S stores $(priv_S, pub_S, pub_U)$, OPRF key k and Env_U
- Login:
  - U runs $F_K(P)$ with S to get R; derives $(priv_U, pub_U)$; gets pub_S, Env_U from S; verifies Env_U
  - U and S run KE with keys $(priv_U, pub_U, priv_S, pub_S)$

OPAQUE with key exchange protocol KE and OPRF $F_K(\cdot)$

- Notes
  - KE: any public-key authenticated key exchange with forward security and KCI security
  - MAC: Needs key commitment property (HMAC yes, GMAC no)
  - OPRF: Secure against malicious participants and output of length ≥ 256 bits
  - OPAQUE proven UC (strong) aPAKE in RO under OMDH

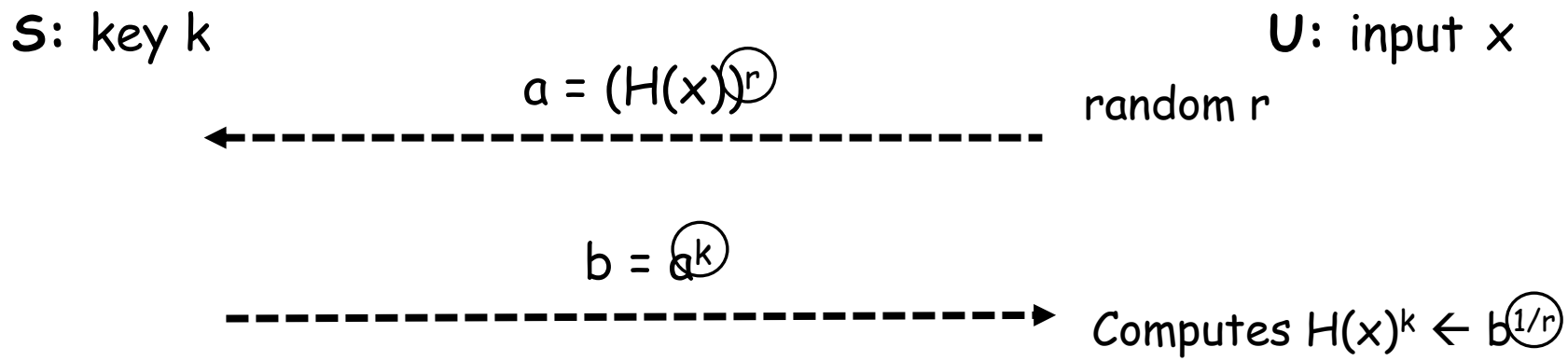
Ok… Nice. But how do you build OPRFs?

OPRF Implementation: DH-OPRF

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

(2HashDH OPRF)

- **PRF**: $\mathbf{F_k(x) = H(x)^k}$ **;** H = RO into group of prime order q; key k in Z_q
- Oblivious computation via Blind DH Computation (S has k, U has x)



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)

Cost: 2 messages; 2 exponentiations for U, 1 for S

OPAQUE with key exchange protocol KE and OPRF $H'(x, H(x)^k)$

- At registration S stores for U: k, $priv_S$, pub_S, pub_U, $Env_U = Mac_R(pub_S)$
- At login:
  - C sends $a = H(P)^r$, g^x (random r, x)
  - S replies with $b=a^k$, pub_S, Env_U , g^y, $Mac_m(g^x, g^y)$ (random y)
  - C sets $R = H'(P, b^{1/r})$, verifies Env_U , derives $priv_U$, verifies Mac, and sends $Mac_m(g^y, g^x)$
- Mac key m and session key derived as prescribed by underlying KE protocol (e.g., 3DH, HMQV, etc.)

- Note: OPAQUE also well suited for TLS 1.3 (with KE = SIGMA)
  - Don't ask what TLS can do for passwords, ask what passwords can do for TLS.

OPAQUE with TLS

- Blends smoothly with 3-flight TLS 1.3 handshake
  - server's cert replaced with $priv_S$ and client's signature uses $priv_U$

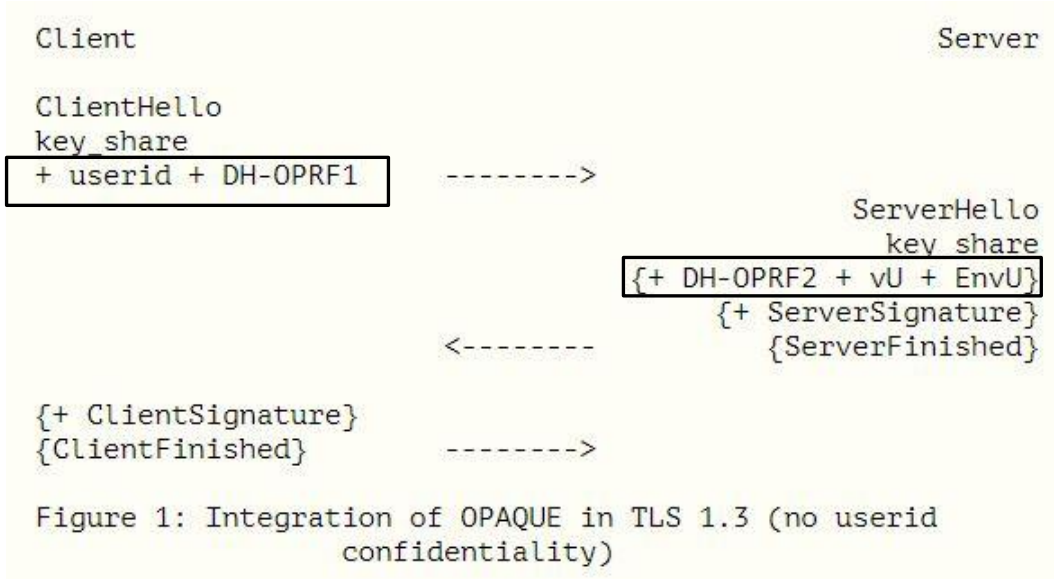


Figure 1: Integration of OPAQUE in TLS 1.3 (no userid confidentiality)

- For user account confidentiality: Adds a round trip, with account info protected by a server-authenticated 1-RTT (with server's cert)

Client-side hardening

- Instead of R = H’(P, $b^{1/r}$) , R=Hardening(H’(P, $b^{1/r}$))
- Hardening: PBKDF2, Scrypt, Argon

OPAQUE and Quantum

- OPAQUE with PQ OPRF and PQ KE is PQ
- What about OPAQUE with DH-OPRF (and PQ KE)?

  What if powerful quantum machine breaks DH in 20 years

  - All an attacker can do is *run an offline dictionary attack*
  - ... And if successful, it learns your 20-year-old password (hopefully you changed it by then)
  - It also assumes the attacker stored communication from 20 years ago (if OPAQUE runs over TLS then attacker needs to also be an active one)
- Lots of work on PQ OPRF – making progress…

OPAQUE for Secret Retrieval

- Goal: Deposit a random key at the server without the server (or anyone) knowing the key
- How? Run OPAQUE and derive the key from R (can use that key to encrypt any other secret)
- Applications of retrieved key:
  - Key to private information (e.g., a backup/file encryption key)
  - E.g., Whatsapp and Facebook Messenger uses it to backup user's keys for encrypting chat history
  - As a key for end-to-end encryption (e.g., in a cloud system)
  - A private key for a crypto wallet (digital assets, identity, etc.)
- Can run OPAQUE in retrieval mode without KE

Summary: OPAQUE Protocol

- **Your choice for password applications**:
  - user authentication/key exchange
  - key/secrets retrieval (in particular, end-to-end security, backups, wallets)
- Efficient, flexible (composes with most KEs)
- Widely deployed: WhatsApp, Facebook Messenger
- Proof of security in a strong UC model
  - Only unavoidable attacks: *online guessing and offline dict. attacks upon server compromise*
  - No reliance on TLS or PKI
  - Password never exposed outside client machine
  - Forces use of secret salt (the OPRF key)

Summary: OPAQUE Protocol

- Standardization: Soon to be RFC
  - **Winner** of CFRG password protocol competition
- Not covered here: Even more security via threshold OPRF
  - Attacker must break multiple hosts before starting an offline dictionary attack
  - Proactive security: Threshold security with implicit rotation (of shares, not keys)

Thank you!! Questions?

hugoaws@amazon.com

hugokraw@gmail.com