The Pythia PRF Service

Adam Everspaugh, Rahul Chatterjee, Sam Scott, Ari Juels, Thomas Ristenpart
Summary

Passwords: Ubiquitous, but vulnerable to offline attack

New direction: Complete architecture for password storage using a new cryptographic PRF service.

Better: no offline attacks, compromise recovery, key management, cryptographic erasure of stolen information
Website stores one of:

- pw
- Hash(pw)
- salt, Hash(salt, pw)
- salt, Hash_{4096}(salt, pw)

6.5M hashes leaked
90% recovered 2 weeks
Password Database Compromises

Password DB breaches are common
Facebook’s Password Onion

```
$cur  = 'password'
$cur  = md5($cur)
$salt = randbytes(20)
$cur  = hmac_sha1($cur, $salt)
$cur  = remote_hmac_sha256($cur, $secret)
$cur  = scrypt($cur, $salt)
$cur  = hmac_sha256($cur, $salt)
```

Archeological record of FB’s struggles with password security.
Facebook’s Password Onion

```
$cur  = ‘password’
$cur  = md5($cur)
$salt = randbytes(20)
$cur  = hmac_sha1($cur, $salt)
$cur  = remote_hmac_sha256($cur, $secret)
$cur  = scrypt($cur, $salt)
$cur  = hmac_sha256($cur, $salt)
```
Remote HMAC Distributes Trust

Hard to detect online attacks

How do we rotate $\text{secret}$?

Web Server

Crypto Server

pw db

hmac($cur, $secret)$

$cur_1$
$cur_2$
$cur_3$

How do we rotate $\text{secret}$?
Our Approach: Pythia PRF

- Detects online attacks
- Cryptographic erasure
- Compromise recovery
- Proactive key rotation
- Multi-tenant

Web Server

Pythia PRF Service

$secret

pw db

Compromise recovery
Proactive key rotation
PRF Query — New User

User

Web Server

Pythia Server

User, pw

t := random()
x := blind(pw)

query: w, t, x

z := unblind(y)

store: (user, t, z)

Web Server ID

Blinded PW

User ID

Protected PW

k := keytable[w]
y := F_k(t, x)
Compromise Recovery

Web Server

Password db is useless

Pythia Server

Doesn’t require original password
User password remains unchanged
Existing Crypto Primitives are Insufficient

Key Updateable Encryption

Deterministic

PRFs

Pseudorandom

Oblivious PRFs

Proxy Re-encryption

Key Rotation

Partially Oblivious PRF

Partially-Blind Signatures

(Partial)
Partially Obl. PRF Construction

Bilinear Pairing

\[ e : G_1 \times G_2 \rightarrow G_T \]

\[ e(a^x, b^y) = e(a, b)^{xy} \]

**Web Server**

\[ x := H(pw)^r \]

\[ \text{blind()} \]

**Pythia Server**

\[ k := \text{keytable}[w] \]

**PRF Query**

\[ w, t, x \]

\[ y := e(H(t), x)^k \]

\[ \text{F}_k(t, x) \]

\[ z := y^{1/r} = e(H(t), H(pw))^{k * r * 1/r} = e(H(t), H(pw))^k \]

\[ \text{unblind()} \]

Similar use of pairings: [Sakai, Ohgishi, Kasahara] [Boneh, Waters]
Partially Obl. PRF Construction

Web Server

Compromise Recovery

Pythia Server

\[ z' := z^{k'/k} = e(H(t), H(pw))^{k*k'/k} = e(H(t), H(pw))^{k'} \]

update()
Advantages of Partially Obl. PRF

- Doesn’t learn secret key
- Detects online attacks
- Doesn’t learn pw
def verify(username, pass):
    (salt, check) = authTableLookup(username)
    digest = hashpass(salt, pass)
    ppass = pythia.query(server, w, t, pass)
    digest = pythia.combine(ppass, digest)
    return digest == check

Easy to Deploy

Small change to code base
No impact on user experience
Parallel Password Onion

Web Server

\[
a = H^{4096}(pw, t) \quad x := \text{blind}(pw) \quad z := \text{unblind}(pw)
\]

Pythia Server

\[
\text{result} := z^a
\]

No performance penalty
Strictly better security — Defense in depth
Pythia Open Source Implementation

nginx  MongoDB

Source code on GitHub

Find links and information at:
https://pages.cs.wisc.edu/~ace

Test + Development server: remote-crypto.io
Fast, Scalable PRF Service

PRF Query: 11.8ms (LAN)  96ms (WAN)

Throughput: 1350 connections/sec  (8-core EC2 instance)
Within factor of 2 of a TLS query

Storage: O(1) per web server
Supports arbitrary number of users for each web server

100M Web Server: 18.6 GB  (keytable)
Beyond Web Servers

File Encryption with remote erasure

Bitcoin Brainwallet

Message-locked Encryption
Conclusion

Password storage is **broken**: too easy to **crack** with offline attacks

**Pythia PRF:**

- prevents offline attacks, **detects** online attacks
- enables compromise **recovery** via key rotation, and crypto **erasure** via deletion
- democratizes access with a **service** architecture