USENIX'18 - Everspaugh - The Pythia PRF Service

提出一种PRF service: PYTHIA, 基础出为 verifiable partially-oblivious PRF, 输入的部分信息为service 已知, 另一部分未知。此类方案, 一方面为了加强存储口令的安全性, 抵抗离线暴力攻击; 另一方面, 通过service 已知的部分信息, 对在线猜测攻击做限制。本方案支持存储值的有效更新; 依赖于 bilinear pairings 及 zero-knowledge proof; 本文给出了形式化定义和安全证明; 在Facebook 和 brainwallet 中应用了此方案.

优点: ① 提出partially-oblivious PRF. 可缓解在线猜测攻击的威胁并防止离线字典攻击
② 文章结构完整, 设计原理, 实现, 与其它方案的对比评估及应用均有描述.
③ 方案支持 key 的更新与删除, 无需用户和其他 client 的参与

问题: ① 未显式给出 threat model, 应考虑注册和登录时的安全性 (注册时应隐藏口令, 如采用 SRP 添加为指数的方法, 否则认证服务器仍可知口令).
② brainwallets 中, t=id 公开给 service, 违背了bitcoin 的匿名性
③ 本文许多表达方式值得学习, 但仍有部分问题例文章对 client, webserver 和 PRF service 的区分仍不够明显, 有些地方 server 不知是 PRF 还是认证服务器

Motivation:

▲最佳实践表明以加盐哈希存储口令是安全的, 但攻击仍可以在得到已丢失的口令时进行暴力破解攻击; 例如 Facebook 的企业利用PRF service, 口令在存储和验证之前需先发送到该服务。- apply a cryptographic function to client-selected inputs under a service-held key。一方面抵抗了离线暴力攻击, 另一方面可通过 rate-limit 检测在线攻击

▲ PRF service 存在的问题: ①在实现中可能存在问题 ②当 client 检测到口令被泄露时, 无法做适当的修补. (理想情况下应允许 cryptographically erase)

▲ PYTHIA 提供了灵活性与可扩展性, 需要密码原语和系统架构方面的创新.

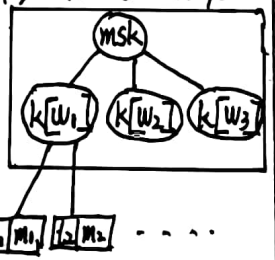
PYTHIA的原理、功能与安全目标. (client 和 web server 指认证 server, PYTHIA 为服务)

▲ PRF的输出为: $Y = F_k(t, m)$, 在口令存储方案中存储于数据库中
- t, m 是每个用户所特有的, m 在口令存储方案中为口令值, 但 t 为公开的
- k: client-specified secret key k held by the service

▲ PRF service 应提供的功能

service 持有 msk, 用于创建 tree, 生成 subkey。从 msk 中可派生许多 ensemble key, 由 client 用于发起 PRF 调用, 通过 k[w] 标记在表中的 ensemble key.

- w: client-chosen ensemble selector → 用于提供 key 的删除和更新操作
- k[w]: ensemble pre-key, client 选择并持有
- ensemble key $k_w = HMAC(msk, k[w])$.



可应用的场景: ① Enterprise deployment ② Public cloud service ③ Public Internet Service.

▲ PYTHIA service 需提供的3个需求: ①低延迟协议(single-round trip 且易于实现, 1调用) ②可扩展到更多应用中 ③输出值应与 r 的随机数不可区分

→基本需求: ① Message Privacy: 保证 m 的私密性, PRF 无法获取信息
② Tweak visibility: PYTHIA 应可见 tweak, 提供 rate limiting 防止在线猜测攻击.
③ Verifiable: client 可验证 PRF service 的输出是否正确
④ Client-requested ensemble key rotations: client 可以更新 k[w]
⑤ Master secret rotations: service 提供的对 msk 的更新
⑥ Forward security: ensemble key 和 master secret key 的更新功能可带来前向安全性

▲设计 PYTHIA 的挑战

① cryptographic: 当前协议无法满足这些要求, 需使用 oblivious PRF 的变体及盲签名的组合的密码原语.
② 创建一个具有 full-featured 的提供核心密码学协议模块的挑战.

Partially-oblivious PRFs (two-party protocol)

PRF-Cl(w, t, m)		PRF-Srv(msk)
$r \leftarrow Z_q$, $X \leftarrow H_2(m)^r$	w, t, X →	$\tilde{X} \leftarrow e(H_1(t), X)$
		$k_w \leftarrow HMAC(msk, k[w])$
		$p_w \leftarrow g^{k_w}$, $y \leftarrow \tilde{X}^{k_w}$
若 p_w match 且 π 验证通过返回 y, 否则返回 ⊥	← p_w, y, π	$\pi \leftarrow ZKP(DLg(p_w) = DL_{\tilde{X}}(y))$ (存储 y)

使用 admissible bilinear pairing $e: G_1 \times G_2 \to G_T$. (素数阶 q 的群)
$H_0: \{0,1\}^* \to G_0$, $H_2: \{0,1\}^* \to G_2$
$k_w \leftarrow Z_p$. 最终输出为: $F_{k_w}(t, m) = e(H_1(t), H_2(m))^{k_w}$

① blinding the message; 通过指数运算进行 m 的隐藏; 通过计算 $y^{\frac{1}{r}}$ 又可以得到获取 y 的信息
② Verifiability: 若 ZKP π 正确, 则表明 PRF 输出正确. (例: prover 取 $v \in Z_q$, $t_1 = g^v$, 计算 $c = H_3(g, pw, \tilde{x}, y, t_1, t_2)$. 计算 $u = v - c \cdot k$. ZKP 为 $\pi(c, u)$. verifier 计算 $t_1' = g^u \cdot p_w^c$, $t_2' = \tilde{x}^u \cdot y^c$, 验证 $c \overset{?}{=} H_3(g, pw, \tilde{x}, y, t_1', t_2')$, 若相等则输出 true.
③ Efficient key updates: PRF-Srv 可以通过给定的 w, 计算新的 $s' = k[w]^{-1}$ 并更新 k_w', 并且发 $\Delta_w = \frac{k_w'}{k_w} \in Z_q$ 给 PRF-Cl, PRF-Cl 可以通过 $F_{k_w}(t,m)^{\Delta_w}$ 的方式更新所有存储的 PRF 值, 无需
▲ reset ensemble key 为认证操作, 需 PRF-SRV 通过带外方式发送 auth token.

Password Onion

Facebook 以序列的方式结合了 Hashing 和 PRF service, $h_3 \leftarrow PRF\text{-}Cl(h_2) = HMAC[SHA\text{-}256](h_2)$, 3点问题: ① 对 pw 的隐藏不够好 ② 更新 key 需要用户的口令 ③ 序列化导致性能不佳

本文提出 updateable, parallelizable password onion

UpPorOnion(w, sa, pw)

z ← PBKDF(pw, sa)

u ← PRF-Cl(w, sa, pw)

h ← u^z

Ret(h, sa)

PRF-Cl 在注册时保存了 (h, sa)，当用户发送 pw' 时

~~需验证~~ UpPorOnion(w, sa, pw') $\stackrel{?}{=}$ h.

由于 $h = u^z = e(H_1(sa), H_2(pw))^{kw \cdot z}$，故可实现 ensemble key 的更新功能

Hardened Brainwallets

每个 Bitcoin 帐户拥有一对公私钥对 (sk, pk)，私钥 sk 用于产生签名，公钥 pk 验证签名并作为帐户标识

Bitcoin 地址 通过 hashing pk 及编码得到 (拥有私钥代表对帐户的控制权)

sk 通过 帐户口令 P 派生得到 sk = SHA-256(P)，仅用 P 来保护 sk，不够多安全 (猜测 P 可得 sk，通过公开地址验证)

→ 引入 PYTHIA，使用 P 来派生 sk

用户选用唯一标识符 id，帐户标识符 acct 及口令 P.

client 发送 (w=id, t=id||acct, m=P) 给 PYTHIA service，得到 $F_{kw}(t, m) = \tilde{P}$.

使用 $\tilde{P}$ 生成 sk

同上，PYTHIA 可以阻止在线猜测攻击，而且由于 acct 是用户特有的，即使两个用户使用相同的口令也可以得到不同的结果 (故无法同时对多个帐户做离线字典攻击).

▲ 即便 PYTHIA service 被破坏，敌手仍需对 brainwallets 做离线暴力攻击.

▲ 可以通过引入门限方法来缓解 Catastrophic failure of PYTHIA 的问题

类似 Shamir 的方案，提供 (k, n) threshold security

Preparation: client 持有 ensemble key selector · w, tweak t 及 password P.

与 n 个 PYTHIA service 计算得 $q_i = PRF\text{-}Cl_i(w, t, P) \bmod p$

client 选择 k-1 阶多项式 $f(x) = \sum_{j=0}^{k-1} x^j a_j$，secret $s = a_0$

client 计算向量 $\Phi = (\phi_1, \cdots, \phi_n)$，$\phi_i = f(i) - q_i$ 并保存 (非 secret)

Recovery: 通过获取 k 个 q_i，client 可以重新生成 S. $f(0) = a_0 = S$.

Security: 通过门限密码学保证了低于阈值的敌手难以发动离线猜测攻击

Verification: 验证 PRF-Srv 的响应很必要，当有 service 被破坏，可以替换一个 service，而不会影响 $a_0 = S$ 的值