CCS'15 Camenisch - Optimal Distributed Password Verification

提出分布式验证口令的高效方法，口令验证为one-round协议，每个服务器在系数阶群中只做一次幂运算；可提供主动安全，防止动态或瞬态的server compromise，只要在两个refresh时间段之间不是所有服务器都被破坏，就可以防止离线字典攻击。在ROM，gap one-more DH假设、UC模型中证明了安全性

优点：① 方案设计和原型实现描述清晰(省略节约协议内容)。
② 较先前方案，本文工作的效率有提升
③ recovery和key refresh可在无交互情况下进行。

问题：① 比起只需t-out-of-n的门限方案，该方案的鲁棒性不足(可能DoS攻击)。
② 没有与其它方案做对比和评估。
③ 描述或格式问题：a)：部分前置知识未介绍或不明确(如Combinatorial secret share)
b)一些图在文中未提及，且顺序阅读友好性较差。

鲁棒性，它要求所有的服务器去验证，但是只是考虑了口令安全性，其他的内容都没有考虑

The problem of offline dictionary attacks when a server is compromised is inherent whenever that single server can test the correctness of passwords
→ 一个解决方法：将验证口令的能力分散到两个或多个服务器中(Ford&Kaliski 2000)。基于此产生的方案包括TPAKE、TPASS(secret sharing)和DCP等

▲ 除了server compromise，如何恢复为另一问题。
recovery from compromise → proactive security(抵抗transient corruptions)

关键建思路

a)将验证口令的能力分散至n个server，login server与后台多个server共同验证
b) recovery和key refresh过程需要访问backup tape，可以无交互进行，且只引入了加法和伪随机数生成器，该过程可以定期执行做预防措施。
c)使用random-oracle-generated blinding factors作用于所有协议消息，使消息不会暴露服务器的key，只有corrupt时会暴露。

所需前置信息：1> Gap One-More Diffie-Hellman 文章基于此假设进行证明
2> Combinatorial Secret Sharing 无交互生成单位元的share。
3> Secure Message Transmission. $\mathcal{F}_{smt}$ 安全消息传输
4> Pseudo-Random Generators 伪随机数生成器
5> Message Authentication Codes MAC(·,·)

安全定义：在UC框架下定义理想函数。

一个协议安全实现了一个理想函数F：
environment Z无法区分它在与真实的协议Π和真实的敌手A交互
还是与 F 和 simulator SIM。

Z输出1的概率定义为 $Real^{\Pi}_{Z,A}(k)$，$Ideal^{F}_{Z,SIM}(k)$。

主要考虑 transient corrupt，不会泄漏server的backup tape，且可以通过recovery恢复

理想函数F的定义 (sid和ssid全局唯一；查找记录未找到就忽略该input)

1. Initialization (INIT, sid)， sid=(S1, S1, ..., Sn, sid') sid'∈{0,1}^k
2. Account Creation Request. LS发起，需要所有server的参与
a) 接口 CREATE LS触发新用户记录创建(setup, ssid, uid, pwd, proceed, finished) 初始为0
b) 接口 PROCEED. 由后台服务器Si调用，来标识继续creation或login的意愿。
所有Si同意，才能继续上述会话(Si同样可以检测可疑行为并终止)。
若LS被corrupt，则全局guesses counter增加(PWDGUESS)
c) 接口 CREATEOK 可被A调用，使setup完成，finished←1。
若LS为honest，则A只能完成Si都同意的record。
反过来，LS可创建无数假冒记录，且在LS恢复honest时，敌手可以使此类帐户无法登录成功(RESULT接口)。
3. Login 验证提供的uid, pwd'是否正确
a) 接口 LOGIN 由LS调用，输入ssid', uid, pwd', 创建一个登录的记录(login, ssid', uid, pwd', proceed')。置为0
b) 接口 RESULT 允许敌手告知(调用Fdpv)LS验证的结果(所有Si同意，执行时才可以)。
若至少一个server被corrupt，或account由corrupt LS创建，则敌手可以使正确的值变错(反过来不可以，为什么?→ Si可能出错，ki≠ki，会出错，而恰好猜对固定值)
若login result由corrupt LS传输，则global guesses counter↓
敌手可选择是否猜测，或通过PWDGUESS接口来猜测
4. Time Out. TIMEOUT接口，允许LS终止Creation或登录的会话
5. (Un)Corruption & Password Guessing ← transient corruption / permanently corrupted
a) 接口 Corrupt 当LS corrupt，敌手可获取所有会话的口令。
若所有server corrupt，Fdpv将guess置为∞，敌手可无限制访问PWDGUESS接口
b) 接口 REFRESH 恢复transient corruption的server，但会删除未完成的setup, login会话
c) 接口 PWDGUESS. 敌手猜存储口令的唯一可能性，由guesses counter表示。

方案 G: 乘法群, 素数阶 q, 2^k, 生成元 g

哈希函数: $H: B_0 \cdot B_1 \cdot B_2 \to G$, $G \cdot C \to \{0,1\}^{2k}$, $B_3 \to Z_q$.

$PRG: \{0,1\}^k \to \{0,1\}^k \times Z_q \times \{0,1\}^k \times \{0,1\}^k$.

$MAC: \{0,1\}^k \times \{0,1\}^* \to T$.

→ INITIALIZATION

LS

输入(INIT, sid), 检查sid内容是否正确

① $0 \le i \le n$, $\to mk_{i,j} \leftarrow_R \{0,1\}^k$ → $(mk_{i,j})_{j=0, j\neq i}^{n}$ → F_{smt} → S_i.

② 对于 $j=1\sim n$, 使用 $PRG(mk_{0,j})$ 生成

$(mk'_{0,j}, \delta_{0,j}, s_{0,j}, \mu_{0,j}) \leftarrow PRG(mk_{0,j})$.

③ $K \leftarrow_R Z_q$, $L \leftarrow g^K$, $PH, BUSY, USED_0 \leftarrow$ empty.

④ $k_0 \leftarrow K + \sum_{j=1}^{n} \delta_{0,j} \cdot \bmod q$.

⑤ $backup_0 \leftarrow (k_0, (mk'_{0,j})_{j=1}^{n}, L, PH)$

$st_0 \leftarrow (k_0, (s_{0,j})_{j=1}^{n}, (\mu_{0,j})_{j=1}^{n}, L, PH, BUSY, USED_0)$.

S_i.

① 对于 $j=0\sim n$, $j\neq i$, 使用 $PRG(mk_{i,j})$ 生成

$(mk'_{i,j}, \delta_{i,j}, s_{i,j}, \mu_{i,j}) \leftarrow PRG(mk_{i,j})$

② $USED_i \leftarrow$ empty.

③ $k_i \leftarrow \sum_{j=0, j\neq i}^{n} \Delta_{i,j} \delta_{i,j} \bmod q$.

④ $backup_i \leftarrow (k_i, (mk'_{i,j})_{j=0, j\neq i}^{n})$.

$st_i \leftarrow (k_i, (s_{i,j})_{j=0, j\neq i}^{n}, \mu_{0,i}, USED_i)$.

Account CREATION

LS.

① input(CREATE, sid, ssid, uid, pwd)

验证ssid, uid是否存在, 存在则abort.

否则 $BUSY[uid] \leftarrow 1$, $USED_0[ssid] \leftarrow 1$.

② $N \leftarrow_R Z_q$, $u \leftarrow H(uid, pwd)^N$,

$c \leftarrow_R Z_q$, $ch \leftarrow C(c)$

存储(uid, pwd, N, u, c)

ssid, u, ch →

③ 查找(uid, pwd, N, u, c)存在 (abort)

更新存储值.

← $V_i, R_{1,i}, R_{2,i}$, ssid

ssid, c (验证challenge) →

ssid, S_i ←

⑤ 若 $g^{s_i} = L^c R_{1,i}$ 且 $U^{s_i} = V^c R_{2,i}$ ⇒ $PH[uid] \leftarrow G(uid, pwd, v)$.

存储PH[uid], 输出(CREATEOK, sid, ssid)

移除与ssid相关的消息

Si

① input (PROCEED, sid, ssid).

验证 $USED_i[ssid]$ 是否未定义, 定义则abort

② $V_i \leftarrow u^{k_i} \cdot B_{2,0}$, $Y_i \leftarrow_R Z_q$.

$R_{1,i} \leftarrow g^{Y_i} \cdot B_{i,1}$, $R_{2,i} \leftarrow U^{Y_i} \cdot B_{i,2}$

存储 (Y_i, ch).

③ 查找 (Y_i, ch), 若 $C(c) \neq ch$ abort.

④ $S_i \leftarrow k_i c + Y_i + B_{i,3} \bmod q$.

移除与ssid相关的消息.

④ $V_0 \leftarrow u^{k_0} \cdot B_{0,0}$.

$Y_0 \leftarrow_R Z_q$, $R_{1,0} \leftarrow g^{Y_0} \cdot B_{0,1}$

$R_{2,0} \leftarrow U^{Y_0} \cdot B_{0,2}$, $S_0 \leftarrow k_0 c + Y_0 + B_{0,3} \bmod q$.

$V \leftarrow \prod_{i=0}^{n} V_i^{\frac{1}{N}}$. $R_1 \leftarrow \prod_{i=0}^{n} R_{1,i}$, $R_2 \leftarrow \prod_{i=0}^{n} R_{2,i}$

$S \leftarrow \sum_{i=0}^{n} S_i \bmod q$.

Login Request

LS.

Si.

① input (LOGIN, sid, ssid, uid, pwd')

check $PH[uid]$ 定义, $USED_0[ssid]$ 未定义.

② $N \leftarrow_R Z_q$., $USED_0[ssid] \leftarrow 1$

③ $u \leftarrow H(uid, pwd')^N$. → ssid, u. →

④ 存储(uid, pwd', N, u).

① input (PROCEED, sid, ssid)

② check $USED_i[ssid] = 1$, 若是, abort.

③ $V_i \leftarrow u^{k_i} \cdot B_{i,0}$.

④ $USED_i[ssid] \leftarrow 1$ 并存储

← ssid, V_i.

⑤ 查找(uid, pwd', N, u). 不存在则 abort

⑥ $V_0 \leftarrow u^{k_0} \cdot B_{0,0}$.

$V \leftarrow T \prod_{i=0}^{n} V_i^{\frac{1}{N}}$

⑦ 若 $PH[uid] = G(uid, pwd', v)$, 则 $pwdok \leftarrow 1$, 否则 $pwdok \leftarrow 0$.

⑧ 输出 (RESULT, sid, ssid, pwdok), 删除与ssid有关的元组 (uid, pwd', N, u)

Timeout

LS ① input(TIMEOUT, sid, ssid). 查找ssid对应的 (uid, pwd, ...)

② 若ssid是未完成的creation, $BUSY[uid] \leftarrow$ undefined., 删除相关信息.

③ 若ssid是未完成的login, 则删除所有ssid的内容.

Refresh

$S_i: G\{Ls = S_0, S_1, \cdots, S_n\}$

$backup_i = (k_i, (mk_{i,j})_{j=0, j\neq i}^{n}, L, PH)$.

对于 $j=0\sim n$, $j\neq i$, 使用 $PRG(mk_{i,j})$ 生成

$(mk'_{i,j}, \delta_{i,j}, s_{i,j}, \mu_{i,j})$

$k'_i \leftarrow k_i + \sum_{j=0, j\neq i}^{n} \delta_{i,j} \bmod q$, $PH' \leftarrow PH$

从Ls0中获取PH'', 对所有 $PH[uid] = \perp$. $PH''[uid] \neq \perp$ 的uid, 执行 $PH'[uid] \leftarrow PH''[uid]$

$backup'_i \leftarrow (k'_i, (mk'_{i,j})_{j=0, j\neq i}^{n}; L, PH'')$

$USED_i$, $BUSY \leftarrow$ empty

$st'_i \leftarrow (k'_i, (s_{i,j})_{j=0, j\neq i}^{n}, \mu_{0,i}, (\mu_{0,j})_{j=1}^{n}, L, PH', BUSY, USED_i)$

↑ 结果

—— LS only

~~~ except LS.
~~~

安全性分析.

- Simulator 作为了一个敌手与函数下交互，内部运行LS, $S_1, \dots, S_n$等server，抵抗真实的敌手.

Initialization SIM生成 blinding seeds 和MAC keys u_i 是真的随机值.

当诚实的LS收到消息，且MAC在密钥下验证正确，但并非S_2发送，此时SIM abort.（反之亦然）

Random Oracles 仿真了 $B_0, \dots, B_3$, C 返回随机值并维护表.

当 random oracle 碰撞时，SIM abort.

Account creation 若仿真 honest LS，SIM要执行查找操作.

↓若corrupt LS则从开始Account Creation.

收到(CREATE)之后，SIM计算 $u \leftarrow g^N$.

若最终 $g^S L^{-c} = R_1$，$u^S \Pi_{i=0}^{n} V_i^{c} = R_2$ 但 $\Pi_{i=0}^{V_i} \neq u^k$，则SIM abort.

而三者都成立，$PH[uid] \leftarrow_R \{0,1\}^{2k}$，并发送(CREATEOk).

当LS corrupt时，为了验证 pw 哈希是否正确，需回答 $\mathcal{O}$ 的询问 $(\cdot,\cdot,v)$

若 $V \neq H(uid,pwd)^k$，则返回随机值；$V = H(uid,pwd)^k$ 时，SIM减少计数器值.（当 guesses<0 时，SIM abort）.

Login corrupt LS 的情况与 Account Creation 类似.

与 honest LS 交互的协议的模拟取决于 uid 创建时 LS 的 honest 与否

- honest：当 S_i 出错时，模拟验证失败.
- corrupt 仿真预测 $\mathcal{O}$ 的输出，概率可忽略.

Corruption 仿 transient corrupt 了 S_i，SIM可以给出对应的 secret key 和状态.

- - - - - - LS，SIM可返回 pw_0，敌手仿可以通过corrupt获取在运行的协议的 pw，可仿真 N'，并存储 pw 哈希.

仿 permanently corrupt 了 S_i，它可以选 mk，并更换 backup tape 的内容.

Refresh SIM可使所有未在PC中的 server refresh.

从 Gap One-More DH归约

仿希望可以使 guesses<0，guesses 初始化=0 { 使用 $V=H(uid,pwd)^k$ 询问 $\mathcal{O}(uid,pwd,u)$ 所有诚实服务器都与一个协议会话. ↑ 同一时段 server 都 corrupt → ∞ }

↓解 gap one-more DH 问题 ↓

↳给定输入 (g,X)，可询问 TCDH, DDH

根据对 Account creation, Login, Corruption, 和 Refresh 的分析，得知若使 guesses<0，需使 CDH 可解.

采用 Pairing 的构造.

G_1, G_2, G_t 为阶为 q 的群，pairing function $e: G_1 \times G_2 \to G_t$.

random oracle: $H \to G_1^{\{0,1\}^k \times \{0,1\}^{|N|}}$, $\mathcal{O} \to \{0,1\}^{2k}$, $B_0 \to G_1^{\{0,1\}^k \times N}$.

其中 $L \leftarrow g_2^k$，Account creation 的协议如下：

LS		S_i
①input(CREATE, sid, ssid, uid, pwd). 检查 PH[uid], BUSY[uid], USED$_0$[ssid]是否已定义 若是，abort		
② BUSY[uid] $\leftarrow$ 1, USED$_0$[ssid] $\leftarrow$ 1		
③ $u \leftarrow H(uid,pwd)^N$	ssid, epoch$_0$, u. →	①input(PROCEED, sid, ssid). 检查 USED$_i$[ssid]=1 或 epoch$_0 \neq$ epoch$_i$, 若是，abort
④存储(uid, pwd, N, u).		
	← ssid, V_i ←	② $V_i \leftarrow u^{k_i} \cdot B_{i,0}$. USED[ssid] $\leftarrow$ 1
⑤查找(uid, pwd, N, u)		③消除 ssid 相关记录.
⑥ $V_0 \leftarrow u^{k_0} \cdot B_{0,0}$, $V = \Pi_{i=0}^{n} V_i$ 方		
⑦验证 $e(v, g_2) = e(H(uid,pwd), L)$; 不成立，则 BUSY[uid] $\leftarrow$ undefined, abort		
⑧存储 PH[uid] $\leftarrow G(uid,pwd,v)$. 输出 CCREATEOk, sid, ssid).		
⑨消除 ssid 相关记录		

⚠ 除了使用智能卡，分析了云平台解决此问题的思路，虚拟机更容易从损坏中恢复.