现代密码学

Modern Cryptography

张方国

中山大学计算机学院

Office: Room 305, IM School Building

E-mail: isszhfg@mail.sysu.edu.cn

HomePage: https://cse.sysu.edu.cn/content/2460

第22讲 数字签名（二）

第7章 签名方案

- ECDSA：

- 可证明安全的签名方案

  一次签名

  FDH-RSA

从DSA到ECDSA

对DSA的批评，其一是NIST没有公开数字签名方案的选择过程，主要是没有工业部门的参与；其二是最初模p的大小固定在512比特，作为答复，NIST对标准作了修改，允许使用不同大小的模。

在2000年，椭圆曲线数字签名算法ECDSA作为FIPS186-2得到了批准。这个签名方案可视为DSA在椭圆曲线情形下的修改。

假设我们有两个定义在$\mathbb{Z}_p$上的椭圆曲线上的点A和B。离散对数$m = \log_A B$是私钥。A的阶是大素数q。

DSA和ECDSA之间的主要差别。在DSA中，值$\alpha^k \mod p$通过模q约化产生的签名(γ, δ)的第一个分量γ。在ECDSA中，类似值是r，r是通过椭圆曲线上的点kA的x坐标模q约化而产生的。该r是签名(r, s)的第一个分量。

ECDSA：签名

设p是一个大素数，E是定义在$\mathbb{F}_p$上的椭圆曲线。设A是E上阶为q（q是素数）的一个点，使得在$<A>$上的离散对数问题是难处理的。

设$\mathscr{P}=\{0,1\}^*, \mathscr{A}=\mathbb{Z}_q^*\times\mathbb{Z}_q^*$，定义

$$K=\{(p,q,E,A,m,B): B=mA\}$$

其中$0\le m\le q-1$。值p,q,E,A,B是公钥，m是私钥。

对于$K=(p,q,E,A,m,B)$和一个秘密的随机数k，$1\le k\le q-1$，定义

$$sig_K(x,k)=(r,s)$$

其中$kA=(u,v)$，$r=u \mod q$，以及$s=k^{-1}(SHA-1(x)+mr) \mod q$

ECDSA：验证

对于$x \in \{0,1\}^*$和$r,s \in \mathbb{Z}_q^*$，验证是通过下面的计算完成的：

$$w = s^{-1} \mod q$$

$$i = wSHA-1(x) \mod q$$

$$j = wr \mod q$$

$$(u,v) = iA + jB$$

$$wer_K(x,(r,s)) = true \Leftrightarrow u \mod q = r$$

密码的分类

- **核心密码：**
  核心密码保护信息的最高密级为绝密级
- **普通密码：**
  普通密码保护信息的最高密级为机密级
  《密码法》规定：核心密码、普通密码属于国家秘密。密码管理部门依照本法和有关法律、行政法规、国家有关规定对核心密码、普通密码实行严格统一管理
- **商用密码：**
  商用密码用于保护不属于国家秘密的信息。公民、法人和其他组织可以依法使用商用密码保护网络与信息安全
  商用密码既可以保护企业商业密码、公民个人隐私，也可以保护政务领域中不属于国家秘密的工作信息。

SM系列

- 国家商用密码算法，简称商密(SM)：是我国自主研发创新的一套密码算法
- 我国公开的国产商用密码算法包括SM1、SM2、SM3、SM4、SM7、SM9及祖冲之算法，其中SM2、SM3、SM4最为常用，用于对应替代RSA、DES、3DES、SHA等国际通用密码算法体系。

对称密码算法	序列密码算法	ZUC
	分组密码算法	SM1、SM4、SM7
非对称密码算法	SM2、SM9	
哈希密码算法	SM3	

SM2

- SM2椭圆曲线公钥密码算法(简称SM2算法)在2010年12月首次公开发布，2012年正式颁布为中国商用密码标准(标准号为GM/T 0003—2012)。2016年8月SM2从中国商用密码标准成为中国国家密码标准（标准号为GB/T32918-2016）。
- 2018年11月，SM2密码算法正式成为ISO/IEC国际标准。
- SM2标准共有四个部分：

  第一部分为总则，必要数学基础知识与相关密码技术，本部分还有四个附录

  第二部分规定了SM2椭圆曲线公钥密码算法的数字签名算法。

  第三部分规定了SM2椭圆曲线公钥密码算法的密钥交换协议。

  第四部分规定了SM2椭圆曲线公钥密码算法的公钥加密算法。

多方签名和门限化

- Yehuda Lindell. Fast secure **two-party ECDSA** signing. In CRYPTO 2017, Part II, LNCS. Springer, Heidelberg, August 2017.
- Yehuda Lindell and Ariel Nof. Fast secure **multiparty ECDSA** with practical distributed key generation and applications to cryptocurrency custody. In ACM CCS 2018. ACM Press, October 2018.
- R. Gennaro, S. Goldfeder, and A. Narayanan, "**Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security**," in ACNS, 2016, pp. 156–174.
- D. Boneh, R. Gennaro, and S. Goldfeder, "Using level-1 homomorphic encryption to improve **threshold DSA signatures for bitcoin wallet** security," in LATINCRYPT, 2017, pp. 352–377.
- Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Two-party ECDSA from hash proof systems and efficient instantiations. In CRYPTO 2019, Part III, LNCS. Springer, Heidelberg, August 2019.
- R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled, "UC non-interactive, proactive, threshold ECDSA with identifiable aborts," in CCS, 2020, pp. 1769–1787.
- G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker, "Bandwidth-efficient threshold EC-DSA," in PKC Part II, 2020, pp. 266–296.
- D. Abram, A. Nof, C. Orlandi, P. Scholl, and O. Shlomovits, "Lowbandwidth threshold ECDSA via pseudorandom correlation generators," IACR Cryptol. ePrint Arch. 2021/1587, 2021

可证明安全的签名方案

- 基于任意单向函数的一次签名方案
- 随机喻言模型下可证明安全的全域哈希签名

一次签名

我们描述一个从单向函数构造一个可证明安全的一次签名方案的概念（如果一个签名方案仅给一则消息签名是安全的，则该签名方案是一次签名方案。当然，该签名可进行任意次验证）。该签名方案，又称为Lamport签名方案。

设k是一个正整数且$\mathscr{P}=\{0,1\}^k$。假定$f:Y\to Z$是一个单向函数，并且$\mathscr{A}=Y^k$。设随机选择$y_{i,j}\in Y$，$1\le i\le k$，$j=0,1$。

设$z_{i,j}=f(y_{i,j})$，$1\le i\le k$，$j=0,1$。密钥K由2k个y和$2k$个z构成。y是私钥，z是公钥。

对于$K=(y_{i,j},z_{i,j}:1\le i\le k,j=0,1)$，定义

$$sig_K(x_1,\ldots,x_k)=(y_{1,x_1},\ldots,y_{k,x_k})$$

关于消息$(x_1,\ldots,x_k)$的签名$(a_1,\ldots,a_k)$验证如下：

$$ver_K((x_1,\ldots,x_k),(a_1,\ldots,a_k))=true\Leftrightarrow f(a_i)=z_{i,x_i}\quad 1\le i\le k$$

一次签名

假设要签名的消息是一个二进制k元组。消息的每个比特都应单独签名。如果消息的第i比特等于j，则签名的第i个元素是$y_{i,j}$，它是公钥值$z_{i,j}$的原像。

验证就是检查签名的每一个元素对应于消息第i比特的公钥元素$z_{i,j}$的原像。这用公开的函数f来完成。

例子：z=f(y)=SHA-3(y)

一次签名

Oscar不能伪造签名，因为他不能求单向函数f的逆以获得秘密y。

然而，该签名方案只能安全地给一则消息签名。在已知两则不同消息签名的情形下，Oscar可构造出与这两则消息不同的消息的签名是很容易的事情。

例如，如果Oscar得到消息(0,1,1)和(1,0,1)的签名，则能对消息(1,1,1)和(0,0,1)签名。

一次签名的安全性

假定$f:Y\to Z$是一个双射单向函数，且公钥包括$2k$个属于Z的不同的元素，Lamport签名方案的安全性是可以证明的。

我们考虑唯密钥攻击：即攻击者只知道公钥。假设攻击者能实现存在性伪造。换句话说，攻击者输出一个消息x和一个有效的签名y（假定f,Y,Z,k是已知的）。

攻击者通过Lamport-Forge算法来模型化。为了简单起见，假设Lamport-Forge是确定性的：给定任何特定的公钥，它总能输出同样的伪造签名。

我们先描述Lamport-Preimage算法，对于随机选择的元素$z\in Z$，该算法找出关于函数f的原像。这个算法是一个使用Lamport-Forge算法作为喻示器的归约。这种归约的存在性与函数f的单向性矛盾。因此，若我们相信f是单向的，则可得出唯密钥的存在性伪造是计算上不可行的结论。

一次签名的安全性

算法7.1 Lamport-Preimage(z)

external f, Lamport-Forge

Comment:我们假定$f: Y \to Z$是双射函数

选择随机值$i_0 \in \{1,...,k\}$和随机值$j_0 \in \{0,1\}$

构造随机公钥$Z = (z_{i,j} : 1 \le i \le k, j = 0,1)$，使得$z_{i_0,j_0} = z$

$((x_1,...,x_k),(a_1,...,a_k)) \leftarrow Lamport - Forge(z)$

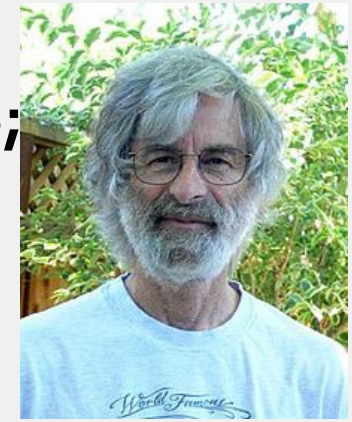
如果$x_{i_0} = j_0$，那么返回(a_{i_0})，否则返回"失败"。

该算法平均成功的概率等于1/2。

定理7.1假设$f: Y \to Z$是一个单向双射函数，并且存在一个确定性算法Lamport-Forge，对于在Z上的包含$2k$个不同元素的任何公钥Z，它能使用唯密钥攻击为Lamport签名方案构建一个存在性伪造签名。那么存在一个算法Lamport-Preimage，使得至少以1/2的概率找出一个随机元素$z \in Z$的原像。

从一次签名构造签名方案

a) 任意单向函数可构造一次签名OTS;
b) 从OTS到多次签名:
"Chain-Based" or "Tree-based"
Merkle tree



NIST Post-Quantum Cryptography：**SPHINCS**

Aumasson, J. P., Endignoux, G.: Design and implementation of a post-quantum hash-based cryptographic signature scheme. NIST Post-Quantum Cryptography(2017), https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions/Gravity-SPHINCS.zip

Bernstein, J. D., Dobraunig, C., Eichlseder, M., Fluhrer, S., et al: SPHINCS+. NIST Post-Quantum Cryptography(2017), https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions/SPHINCS+.zip

全域Hash签名

设k是一个正整数，$\mathscr{F}$是一个单向限门置换族，其中对所有的$f \in \mathscr{F}$，有$f:\{0,1\}^k \to \{0,1\}^k$；并设$G:\{0,1\}^* \to \{0,1\}^k$是一个随机函数。设$\mathscr{P}=\{0,1\}^*$且$\mathscr{A}=\{0,1\}^k$。定义

$$\mathscr{K}=\{(f,f^{-1},G):f\in\mathscr{F}\}$$

给定密钥$K=(f,f^{-1},G)$，f^{-1}是私钥而(f,G)是公钥。

对于$K=(f,f^{-1},G)$和$x\in\{0,1\}^*$，定义

$$sig_K(x)=f^{-1}(G(x))$$

关于消息x的签名$y=(y_1,y_2,\ldots,y_k)\in\{0,1\}^k$验证如下：

$$ver_K(x,y)=true \Leftrightarrow f(y)=G(x)$$

全域Hash签名的安全性

假设$\mathcal{F}$是限门单向置换族而G是一个全域随机喻示器。可以证明全域Hash对于使用选择消息攻击的存在性伪造是安全的；然而，我们只证明对于使用唯密钥攻击的存在性伪造是安全的这一更容易的结果。

假设$\mathcal{F}$是限门单向置换族而G是一个全域随机喻示器。可以证明全域Hash对于使用选择消息攻击的存在性伪造是安全的；然而，我们只证明对于使用唯密钥攻击的存在性伪造是安全的这一更容易的结果。

假设一个攻击者（表示为PDH-Forge)在给定公钥并能获取随机喻示器的情况下能以某个概率伪造一个签名（对于值$G(x)$，它能查询随机喻示器，但是没有具体的算法来计算函数G）。PDH-Forge允许做一定次数的喻示器查询，例如q_h。最终，PDH-Forge以某种概率输出一个有效的伪造签名，用ε表示。

我们构造一个FDH-Invert算法，该算法试图求出随机选择的元素$z_0 \in \{0,1\}^k$的逆，也就是给定$z_0 \in \{0,1\}^k$，我们希望PDH-Invert$(z_0) = f^{-1}(z_0)$。

算法7.2 PDH-Invert(z_0, q_h)

external f

procedure SimG(x)

如果$j > q_h$那么返回“失败”，否则，如果$j = j_0$那么让$z \leftarrow z_0$，否则随机选择$z \in \{0,1\}^k$。

$j \leftarrow j+1$

返回z

main

随机选择$j_0 \in \{1,\ldots,q_h\}$

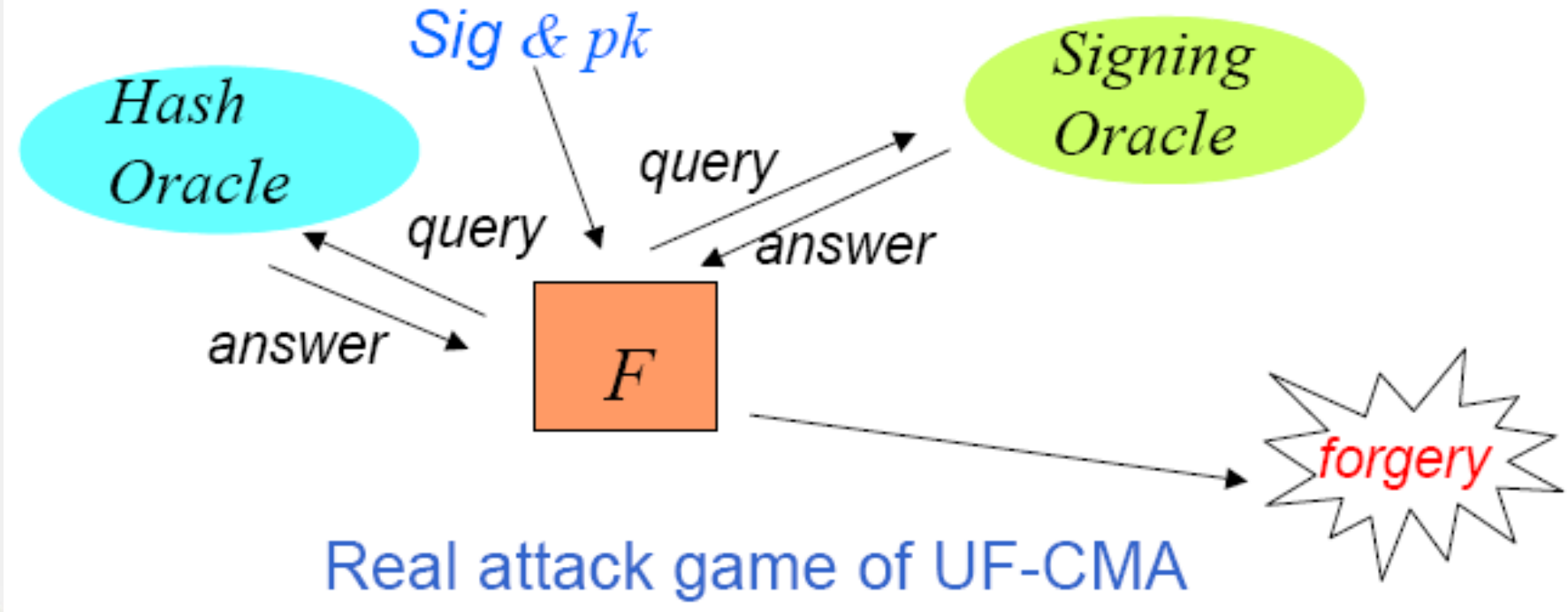
$j \leftarrow 1$

此处插入FDH-Forge(f)代码

如果FDH-Forge(f)=(x,y)，如果$f(y) = z_0$那么返回y，否则返回“失败”。

全域Hash签名

定理7.2 假设存在一个算法FDH-Forge，使用唯密钥攻击，它将以$\varepsilon > 2^{-k}$的概率对全域Hash输出一个存在性伪造签名，那么，存在一个算法FDH-Invert，它将至少以$\frac{\varepsilon - 2^{-k}}{q_h}$的概率找到$z_0 \in \{0,1\}^k$的随机元素的逆。



课外阅读

J.-S. Coron, **On the Exact Security of Full Domain Hash,** M. Bellare (Ed.): CRYPTO 2000, LNCS 1880, pp. 229–235, 2000.

Theorem 2. *Suppose RSA is (t', ϵ')-secure. Then the Full Domain Hash signature scheme is (t, ϵ)-secure where*

$$t = t' - (q_{hash} + q_{sig} + 1) \cdot \mathcal{O}(k^3)$$

$$\epsilon = \frac{1}{(1 - \frac{1}{q_{sig}+1})^{q_{sig}+1}} \cdot q_{sig} \cdot \epsilon'$$

For large q_{sig},

$$\epsilon \simeq \exp(1) \cdot q_{sig} \cdot \epsilon'$$

Proof. Let $\mathcal{F}$ be a forger that $(t, q_{sig}, q_{hash}, \epsilon)$-breaks FDH. We assume that $\mathcal{F}$ never repeats a hash query or a signature query. We build an inverter $\mathcal{I}$ which (t', ϵ')-breaks RSA.

The inverter $\mathcal{I}$ receives as input (N, e, y) where (N, e) is the public key and y is chosen at random in $\mathbb{Z}_N^*$. The inverter $\mathcal{I}$ tries to find $x = f^{-1}(y)$ where f is the RSA function defined by N, e. The inverter $\mathcal{I}$ starts running $\mathcal{F}$ for this public key. Forger $\mathcal{F}$ makes hash oracle queries and signing queries. $\mathcal{I}$ will answer hash oracle queries and signing queries itself. We assume for simplicity that when $\mathcal{F}$ requests a signature of the message M, it has already made the corresponding hash query on M. If not, $\mathcal{I}$ goes ahead and makes the hash query itself. $\mathcal{I}$ uses a counter i, initially set to zero.

When $\mathcal{F}$ makes a hash oracle query for M, the inverter $\mathcal{I}$ increments i, sets $M_i = M$ and picks a random r_i in $\mathbb{Z}_N^*$. $\mathcal{I}$ then returns $h_i = r_i^e \bmod N$ with probability p and $h_i = y \cdot r_i^e \bmod N$ with probability $1 - p$. Here p is a fixed probability which will be determined later.

When $\mathcal{F}$ makes a signing query for M, it has already requested the hash of M, so $M = M_i$ for some i. If $h_i = r_i^e \bmod N$ then $\mathcal{I}$ returns r_i as the signature. Otherwise the process stops and the inverter has failed.

Eventually, $\mathcal{F}$ halts and outputs a forgery (M, x). We assume that $\mathcal{F}$ has requested the hash of M before. If not, $\mathcal{I}$ goes ahead and makes the hash query itself, so that in any case $M = M_i$ for some i. Then if $h_i = y \cdot r_i^e \bmod N$ we have $x = h_i^d = y^d \cdot r_i \bmod N$ and $\mathcal{I}$ outputs $y^d = x/r_i \bmod N$ as the inverse for y. Otherwise the process stops and the inverter has failed.

The probability that $\mathcal{I}$ answers to all signature queries is at least $p^{q_{sig}}$. Then $\mathcal{I}$ outputs the inverse of y for f with probability $1 - p$. So with probability at least $\alpha(p) = p^{q_{sig}} \cdot (1 - p)$, $\mathcal{I}$ outputs the inverse of y for f. The function $\alpha(p)$ is maximal for $p_{max} = 1 - 1/(q_{sig} + 1)$ and

$$\alpha(p_{max}) = \frac{1}{q_{sig}} \left(1 - \frac{1}{q_{sig} + 1}\right)^{q_{sig}+1}$$

Consequently we obtain :

$$\epsilon(k) = \frac{1}{(1 - \frac{1}{q_{sig}+1})^{q_{sig}+1}} \cdot q_{sig} \cdot \epsilon'(k)$$

and for large q_{sig}, $\epsilon(k) \simeq \exp(1) \cdot q_{sig} \cdot \epsilon'(k)$.

The running time of $\mathcal{I}$ is the running time of $\mathcal{F}$ added to the time needed to compute the h_i values. This is essentially one RSA computation, which is cubic time (or better). This gives the formula for t.