Applied Cryptography

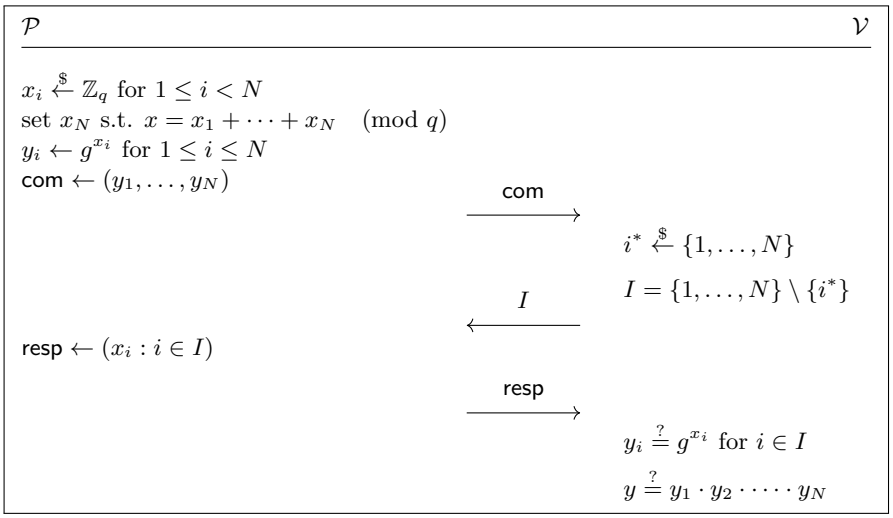
Public Key Cryptography, Assignment 5, Monday, April 22, 2024

Exercises with answers and grading.

1. **(20 points)** The following protocol is at the core of many modern identification schemes and digital signatures. Here, we will try to prove its security and construct a Fiat-Shamir signature out of it with improved efficiency. We will do this through several steps.

   Let p be a prime and g a multiplicative generator of a subgroup G of $\mathbb{Z}_p^*$ of prime order $q|p-1$. And let N be a positive natural number.

   The prover wants to prove knowledge of a discrete logarithm $x \in \{0, 1, \ldots, q-1\}$ of some element y of G, where $y = g^x$. The protocol is as follows:



   (a) Show that the protocol is **complete**.

   (b) Show that the protocol is **sound** and compute the **soundness error**.

   (c) Show that the protocol is **Honest-Verifier-Zero-Knowledge** (HVZK).

   (d) Recall that $x_i \in \mathbb{Z}_q$ and $y_i \in \mathbb{Z}_p^*$. Compute the size of the transcript.

   (e) A common technique to reduce the communication cost is to hash together all the commitments, i.e. set com $\leftarrow H(y_1||\ldots||y_N)$. Modify resp and the verification step so that the protocol remains complete. Recompute the communication cost for this particular case, if we assume usage of SHA-256.

   (f) We can further optimize by first hashing all the y_is separately, and then together as com $\leftarrow H(H(y_1)||\ldots||H(y_N))$. Modify resp and the verification step so that the protocol remains complete Recompute the communication cost for this particular case, assuming again usage of SHA-256. In which case is the communication cost of this version **lower** than in (1e)?

   (g) Using the Fiat Shamir transform, turn the protocol into a digital signature scheme. How many times do you need to repeat the protocol to get a soundness error of 2^{-128}? Write down the signing and verification procedure. For those using LaTeX, we included the LaTeX code for the protocol figure in appendix A.

   (h) Assume now a different version of the protocol above where instead of one i^*, the verifier picks the set I to be of size 1. Write down the new protocol including the two optimizations from (e) and (f).

(i) What is the **soundness error** of this version?

(j) What is the **communication cost** of this version?

(k) Recall from the slides how we can use Merkle trees to improve the efficiency of hash-based schemes. Use Merkle trees to decrease the size of the response and recompute the size of the signature.
Hint: Check again the lecture slides on hash-based signatures.

Begin Secret Info:.....

(a) If the prover knows x, and he constructed x_i and y_i following the protocol then it is easy to see that $y_i{=}g^{x_i}$ for $i \in I$ and $y{=}y_1 \cdot y_2 \cdot \ldots y_N$ both hold, so the verifier will always accept.

(b) If the prover does not know x, he can cheat as follows.
WLOG, he assumes that $i^* = N$ (the same holds for any other assumption for the choice of i^*) Then he prepares as follows.
$x_i \xleftarrow{\$} \mathbb{Z}_q$ *for* $1 \leq i < N$
$y_i \leftarrow g^{x_i}$ *for* $1 \leq i < N$
$y_N = y/(y_1 \cdot y_2 \cdot \ldots y_{N-1})$
and sends the commitment $\mathsf{com} \leftarrow (y_1, \ldots, y_N)$.
He then receives the challenge $I = \{1, \ldots, N-1\}$ *and responds with* $(x_1, x_2, \ldots, x_{N-1})$.
For any other challenge he will fail, because if he would succeed for a challenge including x_N, he could easily reconstruct the secret x.
The soundness error is thus $1/N$.

(c) It is enough to show that the honest Verifier can produce a valid transcript for a given challenge. WLOG, he assumes that $i^* = N$ (the same holds for any other assumption for the choice of i^*) Then he creates the transcript as follows.
$x_i \xleftarrow{\$} \mathbb{Z}_q$ *for* $1 \leq i < N$,
$y_i \leftarrow g^{x_i}$ *for* $1 \leq i < N$,
$y_N = y/(y_1 \cdot y_2 \cdot \ldots y_{N-1})$.
Then: $\mathsf{com} \leftarrow (y_1, \ldots, y_N)$.
The challenge is $\mathsf{ch} = \{1, \ldots, N-1\}$ *and the response* $\mathsf{resp} = (x_1, x_2, \ldots, x_{N-1})$.
The full transcript is then $\mathsf{trans} = (\mathsf{com}, \mathsf{ch}, \mathsf{resp})$.

(d) $|\mathsf{com}| = N\lceil \log_2 p \rceil$, $|I| = (N-1)\lceil \log_2 N \rceil$, $|\mathsf{resp}| = (N-1)\lceil \log_2 q \rceil$. The length of the total transcript is the sum of the three.

(e) The response should include the missing y_{i*} that the verifier can't reconstruct from the other x_i in response. The verification now includes reconstruction of the hash from the commitment and checking against the sent com. *The second check remains the same. The communication cost is now $|\mathsf{com}| = 256$, $|I| = (N-1)\lceil \log_2 N \rceil$, $|\mathsf{resp}| = (N-1)\lceil \log_2 q \rceil + \lceil \log_2 p \rceil$.*
Another valid solution is to notice that y_{i*} can be omitted entirely, in which case this protocol and the one in the next exercise have the same size.

(f) The difference is that the response does not need to include y_{i^*}, but $H(y_{i^*})$. If $\lceil \log_2 p \rceil > 256$ we have a reduction in the communication cost. Then, the verifier can simply recompute y_{i^*} by solving the equation $y = y_1 \ldots y_n$, and the integrity of y_{i^*} can be checked by $H(y_{i^*})$.

(g) *This is a standard application of FS. It is important two note that each repetition of the protocol has two depend on the previous execution somehow. Choose a hash function* $H : \{0,1\} \to \{1,\dots,N\}$*, and let* τ *be the number of repetitions.*

$\mathcal{P}$

$h = \emptyset$ (empty string)
for $j = 0 \dots \tau$ **do** :
  $x_i \xleftarrow{\$} \mathbb{Z}_q$ for $1 \le i < N$
  set x_N s.t. $x = x_1 + \cdots + x_N \pmod q$
  $y_i \leftarrow g^{x_i}$ for $1 \le i \le N$
  $\mathsf{com} \leftarrow (y_1, \dots, y_N)$
  $i^* = H(h, \mathsf{Comm})$
  $I = \{1,\dots,N\} \setminus \{i^*\}$
  $\mathsf{resp}_j \leftarrow (x_i : i \in I)$
endfor
return : $(\mathsf{resp}_1, \dots, \mathsf{resp}_\tau)$

And the verification is as before.

(h) *The new protocol is*

$\mathcal{P}$		$\mathcal{V}$
$x_i \xleftarrow{\$} \mathbb{Z}_q$ *for* $1 \le i < N$		
choose x_N *s.t.* $x = x_1 + \cdots + x_N \pmod q$		
$y_i \leftarrow g^{x_i}$ *for* $1 \le i \le N$		
$h_i \leftarrow H(y_i)$ *for* $1 \le i \le N$		
$\mathsf{com} \leftarrow H(h_1 \| \dots \| h_N)$		
	$\xrightarrow{\mathsf{com}}$	
		$\mathsf{ch} \xleftarrow{\$} \{1,\dots,N\}$
	$\xleftarrow{\mathsf{ch}}$	
$\mathsf{resp} \leftarrow (x_{\mathsf{ch}}, h_i$ *for* $1 \le i \le N, i \ne \mathsf{ch})$		
	$\xrightarrow{\mathsf{resp}}$	
		Compute $H(g^{x_{\mathsf{ch}}})$ *and*
		$\mathsf{com}' \leftarrow H(h_1 \| \dots \| h_N)$
		$\mathsf{com} \stackrel{?}{=} \mathsf{com}'$

(i) *The soundness error is now* $1 - 1/N$.

(j) *The communication cost is now* $|\mathsf{com}| = 256$, $|I| = \lceil \log_2 N \rceil$, $|\mathsf{resp}| = (N-1) \cdot 256 + \lceil \log_2 q \rceil$

(k) *In the protocol above the prover must send the hashes* h_i *for* $i \ne ch$*, which is a total of* $N - 1$*. If on the other hand com was computed using a Merkle trree, then the prover only needs to open the path of ch. That is, every node in the tree that is not a parent of* $h_c h$.

Extra: This is also possible to do for the earlier protocol, where $I = N - 1$*, but it is harder. The trick is to select a random value* **seed***, then use a Merkle tree to generate* $\mathbf{seed}_1, \dots, \mathbf{seed}_N$*, then use these seeds to generate the* x_i*. This way, to send* $N - 1$ x_i *we only need to send* $\log_2(N)$ *seeds, which can be used to reconstruct the Merkle tree, except for the hidden* x_{i^*}.

As a result of this miscalculation this exercise is graded as a bonus.

End Secret Info .....

2. **(15 points)** Recall from Introduction to Cryptography the basics of elliptic curve cryptography. Given an elliptic cure:

$$E : y^2 = x^3 + ax + b$$

over some finite field $\mathbb{F}_p$. We denote by $E(\mathbb{F}_p)$ the points $P = (x, y)$ on the curve (so $y^2 = x^3 + ax + b$ holds) with $x, y \in \mathbb{F}_p$. This $E(\mathbb{F}_p)$ forms a group, so we can add $P_1, P_2 \in E(\mathbb{F}_p)$ together: $P_3 = P_1 + P_2 \in E(\mathbb{F}_p)$. There is also the *point at infinity*, denoted by $\mathcal{O}$, that acts a bit like 0: for any point P we get $P + \mathcal{O} = P$. Adding a point to itself a number of times is denoted by $[n]$, so $[3]G = G + G + G$.

The order of a point is the first natural number n such that

$$[n]G = \underbrace{G + G + \ldots + G}_{n \text{ times}} = \mathcal{O}.$$

Then, we pick a certain point $G \in E(\mathbb{F}_p)$, called the *generator*, which has a large prime order.

(a) As a warm-up before we go to ECDSA: Complete the following toy-example using sage: the parameters will be $p = 43$, $a = 2$, $b = 3$. Making the finite field $\mathbb{F}_p$ and the curve E in sage can be done using

```
F = FiniteField(p)
E = EllipticCurve(F, [a,b])
```

i. Find out the cardinality of the group $E(\mathbb{F}_p)$ using the right command in sage.

ii. Compute the order of the point $G = (11, 25)$ using the right command in sage and explain why this is a good generator. In sage, creating a point on E can be done by `G = E(11, 25)`.

(b) A real-life example is similar but has much larger parameters. We take the well-known and often-used `Curve25519` over a field with prime $p = 2^{255} - 19$. Build the curve E using `E = EllipticCurve(F, [0,486662, 0, 1, 0])` and let the generator G be the point

```
Gx = 9
Gy = 43114425171068552920764898935933967039370386198203806730763910166200978582548
G = E(Gx, Gy)
```

i. Compute the cardinality n of $E(\mathbb{F}_p)$ and the order q of G. What is n/q and what does this value imply?

ii. Alice's secret key a will be
`28111612080006492741872676474874404260743807513041351699201661077095088937 27`
What is her public key?

We will now perform ECDSA on `Curve25519`. ECDSA was discussed in the lectures and is described in the following diagram:

KeyGen:

(a) Let G be a base point of $E(\mathbb{F}_p)$ of order q, where p is an odd prime or a power of 2

(b) Choose a random $d \in \mathbb{Z}_q^*$ and compute $Q = dG$

(c) Output public key $\mathsf{pk} = Q$ and private key $\mathsf{sk} = d$

Sign: Given a message M:

(a) $k \xleftarrow{\$} \mathbb{Z}_q, (x_1, y_1) \leftarrow kG$, and $r \leftarrow x_1 \pmod q$

(b) Set $h = H(M)$ and calculate $s \leftarrow (h + dr)k^{-1}$

(c) Set $\sigma = (r, s)$ and output message-signature pair (M, σ)

Verify: To verify the message-signature pair (M, σ):

(a) Parse $\sigma = (r, s)$, verify $0 < r, s < q$ and calculate $h = H(M)$

(b) Compute $u_1 = hs^{-1} \pmod q$ and $u_2 = rs^{-1} \pmod q$

(c) Compute $X = u_1 G + u_2 Q$

(d) If X is point of infinity, output Fail

(e) Take $X = (x_1, y_1)$, check $r \stackrel{?}{=} x_1 \pmod q$ and output Accept if check succeeds, otherwise Fail

(c) Show the correctness of ECDSA, i.e., that indeed $r = x_1 \mod q$ for valid signatures.

(d) Create a signature (r, s) for $h = 4$ (i.e., compute s). Where the prime p, the curve E, the generator G and the order q of `Curve25519` are the same as above. We also use the same a as secret key for Alice, and the public key P as computed before and for simplicity we have some message M such that $h = H(M) = 4$.
You can use the following code in sage as a help:

```
Fq = FiniteField(q)
h = 4
k = Fq.random_element()
K = int(k)*G //do you see why int() is necessary?
r = mod(K[0], q)
```

(e) Verify that the pair (r, s) is a valid signature for h for the public key Q_b:

```
r = mod(6183170565658761260257204850765575556254837657941527061711072802378821306180, q)
s = mod(9842600004619874760509119930996282404765742930609343408095270094583671540 48, q)
Qb_x = 16261401881118689812438998902523301155067378365752888018839873720289294580865
Qb_y = 85945723076223620328775663133027607450262722542241437425025281261383610244 95
Qb = E(Qb_x, Qb_y)
h = mod(4551593926154650037005279588063523263882387683967338777883944600598235649 2115, q)
```

When building the PlayStation 3, Sony decided to use ECDSA with their private key to sign software. This was done with the goal to separate legitimate software from illegitimate. Sony decided not to take k randomly in the signature generation, but instead to use a static (secret) k for all signatures.

(f) Assume you have two signatures (r_1, s_1) and (r_2, s_2) where k is the same. What do you know about the relation between s_1 and s_2?

(g) Does Sony's decision to take k static impact the security of the signature?

Given the same signature (r, s) as before, the following point $Q_{\text{oh no}}$ will also verify for the same value h:

```
T_x = 32560625091655743179598362635611063129400811572784880556002338716792723350 4
T_y = 25869741026945134960544184956460972567356779614910045322022475500191642319642
T = E(T_x, T_y)
Q_ohno = Qb + T
```

(h) Show that (r, s) also verifies for the point $Q_{\text{oh no}}$.

(i) What is special about the point T here? Why does $Q_{\text{oh no}}$ also work as a public key?

In the Monero cryptocurrency, ECDSA on `Curve25519` is used to create signatures and to sign transactions. In short, users spend their coins by creating a valid signature, and it prevents double spending of a coin by checking if the associated public key has already been used to spend that coin.

(j) Explain how an attacker can spend his coins multiple times using the above special point $Q_{\text{oh no}}$. (This attack was only prevented in May 2017, and it was possible perform this attack for more than three years.)

(k) **(bonus)** How can this attack be prevented?

Begin Secret Info:.....

(a) ***(4 points)***

i. First set up the right curve: `p = 43; a = 3; b = 2; F = FiniteField(p); E = EllipticCurve(F, [a,b])` *Then, the size of* $E(\mathbb{F}_p)$ *is just* `E.cardinality()`*: 19*

ii. Define the point G *on* E *by* `G = E(11,25);` *We compute the order of* G *by* `G.order()`*: 19. So* G *generates every point on* $E(\mathbb{F}_p)$*, and as 43 is prime,* G *is a good generator for ECDH.*

(b) ***(1.5 point)*** *We need to show that* x_1 *of* X *is indeed equal to* $r(\mod q)$*. By writing out we get*

$$\begin{aligned} u_1G + u_2Q &= (u_1 + u_2d)G \\ &= (ms^{-1} + rds^{-1})G \\ &= (m + rd)s^{-1}G \quad \textit{where } s = k^{-1}(m + rd) \\ &= \frac{(m + rd)}{(m + rd)} \cdot kG \\ &= K. \end{aligned}$$

We know that r *is the* x*-coordinate of* K*, and so it equals the* x *coordinate of* $u_1G + u_2Q(\mod q)$*.*

(c) ***(1 point)*** *This depends on the random* k *you sample to compute the signature, but essentially all that is left to do is to compute* s *from the values we have:* `s = k**-1*(m + r*d)`

(d) ***(1.5 point)*** *This code is simply the verification part:*

```
u1 = m*s**-1
u2 = r*s**-1
T = int(u1)*G + int(u2)*Q
r == mod(T[0], q) //prints true
```

(e) ***(1.5 point)*** *When* k *is static, we get* $s_1 - s_2 = k^{-1}(m_1 - m_2)$*. But we know* m_1, m_2, s_1 *and* s_2 *so we can compute which* k *is used every time. Hence, given some* s *and a static* k*, we can compute the secret key* d *by* $s = k^{-1}(m + rd)$*.*

(f) ***(1 point)*** *It completely breaks the security and allows for a recovery of the private key, essentially the worst that can happen to a cryptosystem. To do this simply note that, we have the following equation:*

$$s_1 = (h_1 + dr)k^{-1} s_2 = (h_2 + dr)k^{-1}$$

Therefore:

$$\begin{aligned} &s_1 \cdot {s_2}^{-1} = (h_1 + d \cdot r)k^{-1}(h_2 + d \cdot r)^{-1}k = (h_1 + d \cdot r)(h_2 + d \cdot r)^{-1} \\ &s_1 \cdot {s_2}^{-1}(h_2 + d \cdot r) = (h_1 + d \cdot r) \\ &d \cdot r \cdot (s_1 \cdot s_2 - 1) = h_1 - h_2 \cdot s_1 \cdot s_2 \end{aligned}$$

From which is clear that we can solve for d, which is the secret key:

$$d = (h_1 - h_2 \cdot s_1 \cdot s_2)(r \cdot (s_1 \cdot s_2 - 1))^{-1}$$

There are some edge cases that we are disregarding, such as what happens with if $s_2 = 0$ or $s_1 \cdot {s_2}^1 = 1$, but since this happens with low probability, we can igonore them.

(g) ***(1 point)*** *Simply repeat what we did in (c).*

(h) ***(1.5 point)*** *This special point T has order 8, and so it is on $E(\mathbb{F}_p)$, but not in the group generated by G! It's exactly one of these points in $E(\mathbb{F}_p) \setminus \langle G \rangle$ that we should be worried about. But then also $Q_{oh\ no}$ is on $E(\mathbb{F}_p)$ but not in the group generated by G. However, when computing K, we multiple $Q_{oh\ no}$ by u_2 which happens to a multiple of 8. We get*

$$u_2 Q_{oh\ no} = u_2(Q_b + T) = u_2 Q_b + \mathcal{O} = u_2 Q_b$$

and so the rest of the computation will give a valid signature.

(i) ***(1 point)*** *An attacker can now create multiple valid signatures for the same message m for multiple public keys. This is essentially double spending in such a cryptocurrency.*

(j) ***(+ 1 point)*** *Instead of blindly assuming a public key Q is in $\langle G \rangle$, we check that a public key actually isn't in $E(\mathbb{F}_p) \setminus \langle G \rangle$. We can check this by the order of a point Q, if the order is q, it is in $\langle G \rangle$. So we verify that $[q]Q = \mathcal{O}$.*

(k) ***(+ 1 point)*** *Monero had a market cap of more than 300 million euros when this bug was still feasible, so stealing a million or so using double-spending attacks was doable. Equally so for ZCash and ABCMint. Not many careers give you the option to earn 3 millions in such a short amount of time, hence cryptography is a valid career path. (All of this is modulo market economics.)*

End Secret Info .....

3. **(15 points)** Consider the following generalization of one-time signatures:

KeyGen:

(a) Generate a pair of secret and public key $(\mathsf{sk}_1, \mathsf{pk}_1)$ using Lamport's OTS for 256-bit messages using SHA-256 as the internal hash function H_0

(b) Set a state to $S = ()$ and $\sigma_0 = ()$

Sign: To sign messages M_i, $i = 1, 2, \ldots$ the signer operates as follows:

(a) Generate a new key pair $(\mathsf{sk}_{i+1}, \mathsf{pk}_{i+1})$

(b) Compute $s_{i,\mathrm{OTS}} = \mathsf{Sign}_{\mathsf{sk}_i}(H_1(M_i, \mathsf{pk}_{i+1}))$ where $\mathsf{Sign}_{\mathsf{sk}_i}$ is the signing algorithm of Lamport OTS using the secret key sk_i, and H_1 some hash function

(c) Construct the signature $\sigma_i = (M_i, \mathsf{pk}_{i+1}, s_{i,\mathrm{OTS}}, \sigma_{i-1})$

(d) Add $(M_i, \mathsf{pk}_{i+1}, \mathsf{sk}_{i+1}, s_{i,\mathrm{OTS}})$ to the state S

Verify: To verify the signature $\sigma_i = (M_i, \mathsf{pk}_{i+1}, s_{i,\mathrm{OTS}}, \sigma_{i-1})$:

(a) Check $\mathsf{Vf}_{\mathsf{pk}_j}(M_j, \mathsf{pk}_{j+1}, s_{j,\mathrm{OTS}}) = 1$ for all $j \in \{1, 2, \ldots, i\}$

(a) What should the length of the output of the hash function H_1 be?

(b) Use recursion to compute the length of the signature of the 12-th message. Denote the length of the i-th message by L_{M_i}.

(c) What is the advantage of this scheme compared to MSS introduced in the lectures? What is the disadvantage?

(d) Show that a forgery is possible if instead of $\mathsf{Sign}_{\mathsf{sk}_i}(H_1(M_i, \mathsf{pk}_{i+1}))$ the signer includes $\mathsf{Sign}_{\mathsf{sk}_i}(H_1(M_i))$.

(e) Show that a forgery is possible, if in the signature generation of σ_i, we omit σ_{i-1}, and in the verification process we set $j \in \{i\}$, and provide the OTS public key pk_i together with the signature.

(f) Show that a forgery is possible if the adversary is able to find second preimages for H_1.

Begin Secret Info:.....

(a) The Lamport OTS keys $(\mathsf{sk}_i, \mathsf{pk}_i)$ sign 256-bit messages, so the output of H_1 should be 256 bits.

(b) We sum each individual component for the length of the $i-$th message:

- *the length of M_i is L_{M_i},*
- *the length of pk_{i+1} is $2 \cdot 256 \cdot 256$*
- *the length of $s_{i,OTS}$ is $256 \cdot 256$ again.*

Write L_{i-1} for the length of the previous signature. Then,

$$\begin{aligned} L_i &= L_{M_i} + L(\mathsf{pk}_{i+1}) + L(s_{i,OTS}) + L_{i-1} \\ &= L_{M_i} + 2 \cdot 256 \cdot 256 + 256 \cdot 256 + L_{i-1} \end{aligned}$$

so with $L_0 = 0$, we get $L_{12} = \sum_{i=1}^{12}(L_{M_i} + 3 \cdot 256^2) = 12 \cdot 3 \cdot 256^2 + \sum_{i=1}^{12} L_{M_i}$.

(c) Advantage: You can sign as many messages as you want. Disadvantage: The size of the signature grows with each message

(d) Just change the public key. There is no link that connects the new public key to the signature. The forgery is: $\sigma_i = (M_i, \mathsf{pk}'_{i+1}, \mathsf{Sign}_{\mathsf{sk}_i}(H_1(M_i)), \sigma_{i-1})$

(e) *There is no link to the public key of the signature scheme* pk*, so the adversary can create his own* $(\mathsf{pk}'_i, \mathsf{sk}'_i)$*, and use them to create a signature* $\sigma_i = (M_i, \mathsf{pk}_{i+1}, \mathsf{Sign}_{\mathsf{sk}'_i}(H_1(M_i, \mathsf{pk}_{i+1})))$

(f) *Given a collision* $H_1(M_i, \mathsf{pk}_{i+1}) = H_1(M'_i, \mathsf{pk}'_{i+1})$*, the adversary can make the forgery* $\sigma_i = (M'_i, \mathsf{pk}'_{i+1}, \mathsf{Sign}_{\mathsf{sk}_i}(H_1(M_i, \mathsf{pk}_{i+1})), \sigma_{i-1})$.

End Secret Info .....

A LaTeXcode for Exercise 1 protocol

```
\begin{figure}
    \begin{center}
        \fbox{
        \pseudocode[codesize=\small]{%
        \mathbf{\Prov} \< \< \mathbf{\V} \\[0.1\baselineskip][\hline]
        \> \> \\[-0.5\baselineskip]
        x_i\rand\mathbb{Z}_q \text{ for } 1 \leq i < N\ \> \> \\[-0.2\baselineskip]
        \text{choose\ } x_N \text{ s.t. } %\> \> \\[-0.2\baselineskip]
        x = x_1+\dots+x_N \pmod{q}
        \> \> \\[-0.2\baselineskip]
        y_i \leftarrow g^{x_i} \text{ for } 1 \leq i \leq N \> \> \\[-0.2\baselineskip]
        \com \leftarrow (y_1,\ldots, y_N) \> \> \\[-1\baselineskip]
        \> \sendmessageright*[1.5cm]{\com} \>\\[-0.5\baselineskip]
        % \> \> i_1,\ldots,i_k \rand \{1,\ldots,N\}%, i_j\neq i_k, \forall i\neq k
        \> \> i^* \rand \{1,\ldots,N\}%, i_j\neq i_k, \forall i\neq k
        \\[0.2\baselineskip]
        % \> \> I=
        % \{i_1,\ldots,i_k\}
        % \\[-1\baselineskip]
        \> \> I= \{1,\ldots,N\}\setminus\{i^*\}
        \\[-1\baselineskip]
        \> \sendmessageleft*[1.5cm]{I} \>\\[-0.5\baselineskip]
        \resp\exec (x_i: i\in I)
         \> \> \\[-.2\baselineskip]
        % \text{hashes}=\{h_i: i\notin I\}
        % \> \> \\[-.2\baselineskip]
        \> \sendmessageright*[1.5cm]{\resp} \> \\[-0.5\baselineskip]
        \> \> y_i \stackrel{?}{=}  g^{x_i}  \text{ for } i \in I\\
        \> \> y \stackrel{?}{=} y_1\cdot y_2 \cdot\dots y_N
        }
        }
    \end{center}
\end{figure}
```