MTAT.07.003 Cryptology II
Spring 2012 / Exercise session ?? / Example Solution

Exercise (Liveness proof based on one-way permutation). *Entity authentication protocols are often used to prove liveness of a device or a person. For instance, ATM machines normally ask PIN codes several times during long transactions to assure that the person is still present. Such liveness proofs can be implemented with one-way functions. Let $f : \mathcal{X} \to \mathcal{X}$ be a one-way permutation and let n be the maximal number of protocol invocations. The private key x_n is chosen randomly form $\mathcal{X}$. The public key x_0 is computed by evaluating the following hash chain $x_i \leftarrow f(x_{i+1})$ for $i \in \{0, \ldots, n-1\}$. Each time when a party wants to prove liveness he or she will release non-published sub-key $\hat{x}_i = x_i$. The proof is successful if recompilation of the hash chain $\hat{x}_k \leftarrow f(\hat{x}_{k+1})$ for $k \in \{0, \ldots, i-1\}$ leads to $\hat{x}_0 = x_0$. Prove that (t, ε)-secure one-way permutation protocols are executed sequentially, then the probability that a t-time adversary succeeds in the i^{th} authentication without seeing x_i is at most ε. How large can be the success probability of a t-time adversary that can attack any of these liveness proofs? Show how this protocol can be instantiated with a trapdoor permutation to reduce the size of the private key.*

Solution. Formalisation of the security claim. Let us write a game that models the adversaries attack against the i^{th} protocol instance. By this time the keys $x_1, \ldots, x_{i-1}$ are already released. By taking the most pessimistic view, we can assume that adversary knows these values before the attack against the i^{th} protocol instance. This leads to the following security game:

$$
\begin{array}{l}
\mathcal{G}_0^{\mathcal{A}} \\
\left[\begin{array}{l}
x_n \xleftarrow{u} \mathcal{X} \\
x_{n-1} \leftarrow f(x_n) \\
\cdots \\
x_0 \leftarrow f(x_1) \\
\hat{x}_i \leftarrow \mathcal{A}(x_0, x_1, \ldots, x_{i-1}) \\
\textbf{return } [\hat{x}_i \stackrel{?}{=} x_i]
\end{array}\right.
\end{array}
$$

Security analysis. Recall that a function is (t, ε)-one-way function if $\mathsf{Adv}_f^{\mathsf{ow}}(\mathcal{B}) \leq \varepsilon$ for any t-time adversary $\mathcal{B}$ where the advantage is defined through the following game:

$$
\begin{array}{l}
\mathcal{Q} \\
\left[\begin{array}{l}
x \xleftarrow{u} \mathcal{X} \\
y \leftarrow f(x) \\
\hat{x} \leftarrow \mathcal{B}(y) \\
\textbf{return } [\hat{x} \stackrel{?}{=} x] \ .
\end{array}\right.
\end{array}
$$

Intuitively, it is clear that the ability to find x_i such that $f(x_i) = x_{i-1}$ means that $\mathcal{A}$ can efficiently invert the function. However, we must formally prove this. For that we must first prove that x_i is uniformly chosen from $\mathcal{X}$. Since f is a permutation, the $x_{n-1} = f(x_n)$ is uniformly distributed if $x_n \xleftarrow{u} \mathcal{X}$ and so on. Hence, we can rewite the game $\mathcal{G}_0$ in a simpler form:

$$
\begin{array}{l}
\mathcal{G}_1^{\mathcal{A}} \\
\left[\begin{array}{l}
x_i \xleftarrow{u} \mathcal{X} \\
x_{i-1} \leftarrow f(x_i) \\
\cdots \\
x_0 \leftarrow f(x_1) \\
\hat{x}_i \leftarrow \mathcal{A}(x_0, x_1, \ldots, x_{i-1}) \\
\textbf{return } [\hat{x}_i \stackrel{?}{=} x_i] \ .
\end{array}\right.
\end{array}
$$

Now there is a straightforward reduction construction:

$$\begin{array}{l}\mathcal{B}^{\mathcal{A}}(y)\\ \left\lfloor\begin{array}{l}x_{i-1} \leftarrow y\\ x_{i-2} \leftarrow f(x_{i-1})\\ \cdots\\ x_0 \leftarrow f(x_1)\\ \textbf{return } \mathcal{A}(x_0, x_1, \ldots, x_{i-1})\ .\end{array}\right.\end{array}$$

It is straightforward to verify that by substituting the definition of $\mathcal{B}$ into the game $\mathcal{Q}$, we get a game

$$\begin{array}{l}\mathcal{G}_2^{\mathcal{A}}\\ \left\lfloor\begin{array}{l}x \xleftarrow{u} \mathcal{X}\\ y \leftarrow f(x)\\ x_{i-1} \leftarrow y\\ x_{i-2} \leftarrow f(x_{i-1})\\ \cdots\\ x_0 \leftarrow f(x_1)\\ \hat{x}_i \leftarrow \mathcal{A}(x_0, x_1, \ldots, x_{i-1})\\ \textbf{return } [\hat{x}_i \stackrel{?}{=} x_i]\ .\end{array}\right.\end{array}$$

that is semantically equivalent to $\mathcal{G}_1$. Consequently, we have proven

$$\Pr\left[\mathcal{G}_0^{\mathcal{A}} = 1\right] \leq \mathsf{Adv}_f^{\mathsf{ow}}(\mathcal{B})\ .$$

Since the running time of $\mathcal{B}$ exceeds the running time of $\mathcal{A}$ by $\mathrm{O}(n)$ steps, we can conclude that $(t + \mathrm{O}(n), \varepsilon)$-one-way permutation is sufficient to guarantee that any t-time adversary can succeed at most ε in the attack against i^{th} protocol instance. As there are n protocol instances to attack, the overall probability that a t-time adversary succeeds against any of the protocols is $n\varepsilon$.

Practical implementation. Let us initialize the construction with $f(x) = x^2 \mod N$, where $N = pq$ is RSA modulus. This construction is guaranteed to be one-way function, as long as factoring RSA moduli is computationally intractable. A naive implementation of the scheme would lead to a secret key that consists of $x_1, \ldots, x_n$. However, we can do much better.

First, note that $x_0, \ldots x_{n-1}$ must be a quadratic residues by construction. Second, note that we can efficiently compute square roots if we know the factorisation of N. Thus, we could in principle find x_{i-1} directly from x_i. However, there are four possible square roots and we must choose the correct candidate. If factors $p, q \equiv 3 \pmod 4$ then it easy to prove that either a or $-a$ is a quadratic residue in $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$ and thus also the sets of quadratic residues and quadratic non-residues are equal. Due to the Chinese reminder theorem $\mathbb{Z}_N \simeq \mathbb{Z}_p \times \mathbb{Z}_q$ and we can consider $(x_1, x_2) \in \mathbb{Z}_p \times \mathbb{Z}_q$ instead of $x \in \mathbb{Z}_N$. Now if (x_1, x_2) is a quadratic residue then there are four square roots

$$(\xi_1, \xi_2), (-\xi_1, \xi_2), (\xi_1, -\xi_2), (-\xi_1, -\xi_2)$$

for some $\xi_1^2 = x_1$ and $\xi_2^2 = x_2$. For obvious reasons, the root is a quadratic residue only if the first component is quadratic residue and the second component is quadratic residue. Under our assumptions either ξ_i or $-\xi_i$ is a quadratic residue and thus only one of the four square roots is a quadratic residue. As a result, we have shown that if x_0 is computed from x_n, we can reliably reconstruct all intermediate values x_i by selecting the right square root of x_{i+1}. Moreover, we do not have to select x_n and compute x_0 it is sufficient if we choose x_0 as a quadratic residue and compute x_i on the fly. As x_0 and n uniquely determines x_n, the correspondence must be a permutation. Thus a randomly chosen quadratic residue x_0 assures that x_n is uniformly distributed over quadratic residues, too.

Practical implementation. Another way to define a trapdoor permutation is to use RSA permutation. For that we must first generate RSA modulus N and then a public encryption exponent e, which defines a

trapdoor permutation $f(x) = x^e \mod N$. By knowing the secret decryption exponent d, it is easy to invert the permutation: $f^{-1}(y) = y^d \mod N$. Note that there is no direct reduction which converts RSA inverter into factoring algorithm. However, this problem is considered intractable for randomly chosen encryption exponent. Thus, we can use also this variant although its concrete security guarantees are worse compared to the contraction based on squaring (discussed above).