Applied Cryptography: Assignment 4

Group number 16

Elwin Tamminga s1013846

Lucas van der Laan s1047485

1

(a) It is necessary that the cryptosystem is not based on a hard computational assumption.

(b) Because hard computational problems can be solved by an adversary in possession of a quantum computer using algorithms like Shor's or Glover's. It is assumed that such a quantum computer will exist within our lifetime.

(c) We can use normal pre-quantum CPU cores to still do quantum-safe en-/decryption, instead of everyone needing a quantum computer to be safe. In the beginning most people will not be capable of having a quantum computer, so it would not be safe if an adversary did have a quantum computer, thus we use post-quantum.

(d) Below we list the information for the combinations.

- All lower case: 26
- All upper case: 26
- All ASCII special characters: 33
- All together: 85
- Amount of characters: 10
- Possible combinations: 85^{10}.

Grover's algorithm complexity: $O(\sqrt{N})$

From the slides: on average $\sim$ 4.13 years for 26^8 combinations for a classical attack, so $4.13/26^8$ years per combination.

$4.13/26^8 \cdot \sqrt{85^{10}} \approx 32$ days.

Thus a quantum adversary can break the password in $\sim$ 32 days on average.

(e) $4.13/26^8 \cdot 85^{10} \approx 389\ 361\ 068$ years.
Thus a non-quantum adversary can break the password in $\sim 389\ 361\ 068$ years on average.

(f) The required key length is double the length of that against a normal adversary, thus the required length is 256 bits. This is because Grover's algorithm can do a quadratic speedup for key searching which makes a key of 256 bits have a security strength of $2^{256/2} = 2^{128}$ which leads to a security strength of 128.

(g) The security of RSA depends on integer factorization. The complexity of Shor's algorithm is $O((\log N)^2(\log\log(N))(\log\log\log(N)))$. So to get 128 bits of security against quantum adversaries, we use $N = 2^x$ and solve x in the following equations:
$\log_2((\log 2^x)^2(\log\log(2^x))(\log\log\log(2^x))) = 128$.

This results in $x \approx 2.12935 \cdot 10^{18}$, which is the required length of RSA keys against quantum adversaries for 128 bits of security.

The calculation can be found on this url: `https://www.wolframalpha.com/input?i=log2(log(2^x)^2*(log(log(2^x)))*(log(log(log(2^x)))))+=+128+solve+x`

(h) The learning with errors (LWE) problem relies on finding an unknown linear function f, such that the output of the function is an integer y in the modulo group $\mathbb{Z}_q$ and the input is an integer x in $\mathbb{Z}_q^n$ which denotes the set of n-vectors over $\mathbb{Z}_q$, with high probability $y = f(x)$.
This can be used in a public key cryptosystem by defining a linear function using a secret variable $s \in \mathbb{Z}_q^n$, which will be the private key. A public key can be derived from this key using uniformly and independently chosen variables in combination with s. Values encrypted with the public key can then be decrypted using the secret in the linear function, the output depends on the probability of $y = f(x)$.

2

(a) H_0 is SHA-256, thus the output length is 256 bits.

(b)
- The output length of the Sign function is 65536 bits (256x256), as it is a Lamport signature function.
- The size of the public key pk_i is 512 numbers of 256 bits, thus 131072 bits.

In σ_{12} we have all the previous signatures as well, thus we have a signature that looks like $(M_i, pk_{i+1}, \{0,1\}^{65536}, \sigma_{i-1})$. Thus the length of $\sigma_{12} = \sum_{i=1}^{12}(L_{M_i}) + 12 \cdot (131072 + 65536) = \sum_{i=1}^{12}(L_{M_i}) + 2359296$ bits.

(c) **The advantage:**
The Merkle Signature Scheme is a stateful signature scheme, so used signatures must be tracked, while in this scheme the state is kept in the signatures.

The disadvantage:
The Merkle Signature Scheme uses a binary tree, so the previous hashes do not need to be included in the signature. This makes it much more efficient than Lamport OTS.

(d) We can create a forgery by reusing pk_1. First we do one online query with M_1 which returns σ_1. Then we can create a forgery with $M_2 = M_1$ by replacing pk_2 with pk_1 in σ_1, and create our own valid σ_2 using $\sigma_{2,OTS} = \sigma_{1,OTS} = \text{Sign}_{\text{sk}_1}(H_0(M_1)))$.

(e) We can create a forgery by generating our own key pair (sk_2, pk_2) and (sk_3, pk_3), then we can create a forgery for any message M_2 using $\sigma_2 = (M_2, pk_3, \text{Sign}_{\text{sk}_2}(H_0(M_2, pk_3)))$.
We can send this together with pk_2 for verification, the signature will be valid:
$\text{Vf}_{\text{pk}_2}(M_2, \text{pk}_3, \sigma_{2,OTS}) = 1$.

(f) Given a valid signature σ_i, We can create a forgery $\sigma_i{}'$ for a message M_a by replacing M_i by M_a and finding a second preimage such that $H_0(M_i, pk_{i+1}) = H_0(M_a, pk_{i+1})$.

(g) Instead of σ_{i-1}, we include the signature of the root hash of a Merkle tree. The improved scheme is as follows:

- **Key generation:**
  Stays the same, instead of sk_1 and pk_1 there will be just one keypair (sk, pk).
- **Sign:**
  To sign a message M_i, $i = 1, 2, \ldots$ the signer
  - Computes $h_i = H_0(M_i, pk))$ and expand the Merkle tree stored in S. The parent nodes are recursively updated by computing $H_0(h_{c1}||h_{c2})$ (or $H_0(h_{c1})$ if there is only one child) with h_{c1} and h_{c2} being the children of the parent node.
  - Computes a signature $\sigma_i = (M_i, \text{Sign}_{sk}(h_{root}), \text{merkle proof})$ after adding the hashes of all messages to the tree, with h_{root} being the root hash of the tree and the merkle proof contains all other hashes required to verify if M_i is in the tree.
- **Verify:**
  To verify a signature $\sigma_i = (M_i, \text{Sign}_{sk}(h_{root}), \text{merkle proof})$ with $OTS = \text{Sign}_{sk}(h_{root})$
  - Check $\text{Vf}_{\text{pk}}(M_i, OTS, \text{merkle proof}) = 1$ by checking the OTS and then calculating hashes of $H_0(M_i)$ in combination with the other hashes in the proof to calculate and verify h_{root}.

This speeds up verification, because the verifier only needs to check a part of the hash tree to verify if the message is included in the tree, instead of checking all signatures of all previous messages. This also speeds up signing because only one keypair is required to sign only one time after hashing all the messages.

3

(a) **Attack 1:**
Alice doesn't like Eve and wants to send her harmful messages via an online messaging service. Eve can do a Man-in-the-middle attack on that service, so as a revenge Eve forwards it to Alice's best friend Bob instead, while making Bob think he's talking to Alice.

Attack 2:
Both Eve and Alice want to buy something from Bob. Eve is a Man-in-the-middle between Alice and Bob. Alice tries to authenticate with Bob, so Eve forwards Alice's messages to Bob, but makes Bob think that he is talking to Eve instead. The information on the purchase on Bob's end is Eve's information, but the authentication happens with Alice. Now Alice has bought an item from Bob, but the information on the purchase was Eve's. Eve now has the item that both Alice and Eve wanted, without ever having to pay for it.

(b) There are no identities sent in clear, so these cannot be replaced by Eve. There are only four variables sent over the protocol that a man-in-the-middle (Eve) can replace.

Eve can try to replace A by G^e with e being the private key of Eve to make Bob think he is talking to Eve. However, this will not work because when Eve forwards the identity of Bob together with C_B, then Alice can see that the identity has been replaced by checking $\text{Sign}_{sk_B}(E, B)$ which cannot be replaced by Eve because it does not have the signing key of Bob.

Eve can also try to replace B by G^e to make Alice think she is talking to Eve. However this will also not work for the same reason as above, because Eve cannot replace C_A as it includes $\text{Sign}_{sk_A}(E, A)$.

(c) Eve can just replace the clear identity "Alice" by "Eve". Bob will then send $\text{Sign}_{sk_B}(A, B, Bob)$ which Eve can forward to Alice to make her believe she is talking to Bob. Because Bob now thinks he is talking to Eve, Eve can just replace the signature sent by Alice to Bob by $\text{Sign}_{sk_E}(A, B, Eve)$, which will be valid because Bob will verify it using the public key of Eve.

(d) Eve can still replace the clear identity "Alice" by "Eve". Bob will then send $\text{Sign}_{sk_B}(A, B, K_{B,A})$ which Eve can forward to Alice to make her believe she is talking to Bob. Because the protocol uses RSA signatures, the plaintext of the signature is known, thus Eve now knows $K_{B,A}$. Because Bob now thinks he is talking to Eve, Eve can just replace the signature sent by Alice to Bob by $\text{Sign}_{sk_E}(A, B, K_{B,A})$, which will be valid because Bob will verify it using the public key of Eve.

Removing the identities in clear from the protocol does nothing, Eve can still replace the signatures to do an identity misbinding attack, and there is no other way to verify who they are talking to.