DD2448 Foundations of cryptography

Homework II
krypto17

Persnr	Name	Email
951113-7656	Artem Los	arteml@kth.se
760513-7319	Mati Rachamim	rachamim@kth.se
-	- -	-

1

1a (1T) In the literature pertaining to the RSA algorithm, it has often been suggested that in choosing a key pair, one should use so-called "strong" primes p and q to generate the modulus n. Strong primes have certain properties that make the product n hard to factor by specific factoring methods; such properties have included, for example, the existence of a large prime factor of p-1 and a large prime factor of p+1. The reason for these concerns is that some factoring methods - for instance, the Pollard p-1 and p+1 methods - are especially suited to primes p such that p-1 or p+1 has only small factors; strong primes are resistant to these attacks. Strong primes are required in for example ANSI X9.31

However ,advances in factoring over the last ten years appear to have obviated the advantage of *strong primes*; the elliptic curve factoring algorithm is one such advance. This method is a strict improvement over Pollard's *p-1* method in finding small factors as it allows one to "retry" if a "bad" group order (that is *p-1*) was encountered and hence we can use multiple machines in parallel which in the end will shorten our attack time by the same factor . The new factoring methods have as good a chance of success on strong primes as on "weak" primes. Therefore, choosing traditional "strong" primes alone does not significantly increase security as one can try to generate another elliptic curve which will be smooth. Choosing large enough primes is what matters. However, there is no danger in using strong, large primes, though it may take slightly longer to generate a strong prime than an arbitrary prime.

As written in Rivet's article: "Choosing a strong prime is like locking one door and leaving all the other doors unlocked, generating large primes is like locking all doors"

It is possible that new factoring algorithms may be developed in the future which once again target primes with certain properties. If this happens, choosing strong primes may once again help to increase security.

more details can be found in Rivet's article .

1b (1T) Strong RSA Assumption was introduced by Bari c and Pfitzmann [1] and by Fujisaki and Okamoto [2] This assumption differs from the RSA Assumption in that the adversary can select the public exponent e. The adversary's task is to compute, given a modulus n and a cyphertext C, any plaintext M and (odd) public exponent $e \geq 3$ such that C=M^e(mod n). This may well be easier than solving the RSA Problem, so the assumption that it is hard is a stronger assumption than the RSA Assumption. The Strong RSA Assumption is the basis for a variety of cryptographic constructions.

1c (1T)

1d (1T)

1e (1T)

1f (1T)

1g (1T)

1h (2T)

1i (3T)

1j (2T)

2

2a (2T) Discrete logarithm in $A_q = (Z_q, +)$ is quite easy, since the group operation is addition, "exponentiation" α^a is multiplication by $a \bmod n$. Therefore, we look for integer a such that $\alpha a = \beta \bmod n$, i.e. $a = \log_\alpha \beta = \beta\alpha^{-1} \bmod n$. We can find a by using the *extended euclidean algorithm* defined on p. 156 in the course literature. The algorithm has $O(\log n)$ running time, which is very bad for cryptographic applications (in crypto, we want to have in the order of $O(2^n)$).

[1] `https://people.csail.mit.edu/rivest/RivestKaliski-RSAProblem.pdf`
[2] `https://people.csail.mit.edu/rivest/RivestKaliski-RSAProblem.pdf`

2b (3T) The security level in the problem is way to big as it is usually in the range of $128-256$. Since asymmetric cryptography is based on problems that we think are hard solve, many organisations have different suggestions for the size of parameters.

For the multiplicative group (this is equivalent to considerations for *Diffie-Hellman Key Agreement Method*), we found a resource[3] that summarizes the security level vs. the security parameters in the table below:

System requirement for attack resistance (bits)	Symmetric key size (bits)	RSA or DH modulus size (bits)	DSA subgroup size (bits)
70	70	947	129
80	80	1228	148
90	90	1553	167
100	100	1926	186
150	150	4575	284
200	200	8719	383
250	250	14596	482

The size of the modulus p should be more than 4575 bits and the size of the subgroup q 284 bits to get a 128 bits security level. NIST recommends 3072 bits in the modulus in RSA and since the discrete logarithm algorithms for Diffie-Hellman are similar to factoring large numbers (for RSA), this suggestion applies to Diffie-Hellman as well.

For elliptic curve groups, situation looks better since there is no *index calculus* approach that applies to elliptic curves, so for now solving discrete logarithm for the general case is the best we can get (aka *generic algorithms*). These algorithms have a lower bound of $\Omega(\sqrt{n})$, so for 128 bits of security, we need 256 bit modulus s. The same is for the r (in fact it's actually based on this element, since c is small). In the elliptic curve group, we typically want to have $q = \#E/h, h \in \{2, 4\}, \#E \approx p$ (we are basing assumptions on the finite field F_{2^n} and we can always express 2^n as $c^r * t$) so the number of bits of q is roughly the same as for p or different by a factor of $1/2$ or $1/4$. Note, for elliptic curves, the subgroup can be considerable smaller since there's no known way of applying an index calculus based approach.

2c (2T) Both the discrete logarithm for the multiplicative group and elliptic curve group can be solved in polynomial time using Shor's algorithm (see his original publication, section 6 `http://epubs.siam.org.focus.lib.kth.se/doi/pdf/10.1137/S0036144598347011`).

[3] `https://www.ietf.org/rfc/rfc3766.txt`

2d (3T) The first thing is that the relationship $p = 2q+1$ has to be satisfied for ElGamal to be secure (see p.272 in course literature). The idea is that knowledge about quadratic residue can be exploited to retrieve the private key. The second problem is that ElGamal has the problem of *small subgroup attack*. ECC should be chosen instead, partly because the best attacks are targeting the generic discrete logarithm problem (index calculus cannot be used) and so the order q can be smaller. So, ElGamal groups are vulnerable to index whereas ECC requires polland rho algorithm (which is much slower).

2e (3T) We can convert the sequence of bits to to a number by viewing them as a binary number, so $\{1, 1\} \rightarrow 3$. This is possible since the order $|G_q| = q$ and each message we get is within that range. We could potentially use Huffman encoding if we know the messages sent in advance. Otherwise, it's difficult to see any better way of doing this, since each bit carries information and we have to store it as a value, which we in this case can only do in unbalanced binary representation.

3 (4T) Lamport's schema is a provable-secure signature algorithm which is believed to resist the introduction of quantum computers (this is not the case for RSA_DSA and EC_DSA, for example). As shown in the course literature (p. 302), the security of Lamport's schema depends on the one-way property of the hash function. So, existential forgery attack (when an adversary finds a pair (m, σ) for some m, without knowing (m, σ) in advance) is computationally infeasible if we believe that the hash function is one-way. A proof is available on p. 302, but we can prove this informally too. Let's assume there's a method that can find (m, σ) for some message m. To make this possible, we need to compute the pre-image (i.e. invert the hash function and get the private key), which contradicts that the hash function is one-way. The only approach left is to find the pre-image by brute force, which is computationally hard. SHA256 is assumed to be one-way. Therefore, an existential forgery attack is not feasible assuming that SHA256 is one-way.

4 (12I) Complete

5 (7I) Complete

6 (10I) Complete

7

7a (1T) The controversy was in the constants that the standard did not explain how they were obtained.

7b (1T) First, the PRG had a small bias, which was not a seen as a problem but more a concern, or like Bruce Schneiner put it

> Cryptographers are a conservative bunch: We don't like to use algorithms that have even a whiff of a problem.

Even before Dan Shumow and Niels Ferguson showed the exploit, the point of concern was the fact that the constants were not explained and possibly the fact that it was recommended by NSA (especially after the DES affair).

7c (2T) Inspired by *Dual EC DRBG* malicious attempt to implement a back-door (at least so we think) by specifying constants without an explanation, people started to question both the security of NIST curves such as **P-256** (where not explanation of constants is given) and the use of pre-computed constants where the choice of constants is well motivated by some cryptographic property (such as **Curve25519**). The choices of constants in many curves (as in other cryptosystems) is in many times subjective to the opinions of experts. For example, which PRG for the generation constants should be chosen, which seed should the PRG and many other decisions that have to be made.

The idea behind the *Million dollar curve* is motivated by the fact that *you should not put all the eggs in the same basket*. So, even if **Curve25519** will be preferred instead of **P-256**, there might be a 0-day attack (ie. an attack that is not known at the time) on **Curve25519** which we might know about. Some choices of constants in **Curve25519** (such as the prime) can be discussed.

The million dollar curve is an attempt to build up a cryptosystem (in this case, parameters to a curve) by making the selection of cryptographic settings (eg. curve type, parameters, etc.) at random. The choice of parameters is left to chance and not an individual. The authors behind the million dollar curve suggest to use national lottery result as a seed value to Blum Blum Shub PRNG. They argue that these results are not easily tempered by any one entity. So, essentially, they want to achieve a notion of *public verifiable randomness*, i.e. a randomness that can be verified at a later point in time (ie. we can verify t_o given that $t \geq t_0$).

`http://cryptoexperts.github.io/million-dollar-curve/specifications/2016-02-01_trap-me-if-you-can.pdf`

8 (4T) Intuitively, it does not seem that this can work since we are taking n bits input and map it to a $\log n$ bits output. For PRG, it is obvious that this would not work since the purpose is to use a random seed to get a longer random looking number, so this function would reduce the length of the output, which does not allow us to generate a longer pseudo-random number. However, for a PRF, any input will map to a random looking output. So, one way that we could construct this PRF ($\{0,1\}^n \to \{0,1\}^n$) is to reuse output from multiple calls to $f_{n,\gamma}$. This does seem very strange, since in $f_{n,\gamma}$ we store $2^{\log n} n$ numbers, whereas our new function $f'_{n,\gamma}$ requires $2^n n$ numbers. However, it is possible if we append $n/\log n$ outputs from $f_{n,\gamma}$ to get $f'_{n,\gamma}$. Here we assume that we know the all the outputs from $f_{n,\gamma}$. Sure, we will get collisions (i.e. multiple inputs map to the same output), but if we assume that the original function is a PRF (which also has collisions, in fact $n/\log n$ of them), then we can claim our new function is also a PRF.

9 (3T) Let SHA-256 be random function (no matter what input we have, the output will appear random) defined as $f_{sha256}(x, s)$. x is any value and s is the random seed that is secret. In practise, this function would be defined using HMAC-SHA256 (see `rfc2104`). Based on the the lecture notes (slide 271), we can combine random functions to get a PRG, as shown below:

$$PRG(s) = (f_{sha256}(1, s), f_{sha256}(2, s), \ldots, f_{sha256}(t, s)) \tag{1}$$

Since we need 3000 bits, we would set the value $t = 12$ so that we get a bit more than 3000 pseudo-random bits. Note, we are not done since we want to construct a pseudo-random function (PRF) and not a PRG.

In the lecture notes (p. 272), the PRG should output $2k$ output to get a PRF of k output. Therefore, $t = 25$. Now, let $x_{[i]} = (x_0, \ldots, x_i)$. To compute the new PRF, we do the following:

1. Compute $(r_0^0, r_1^0) = PRG(s)$
2. Compute $r^i_{x_{[i]}||0}, r^i_{x_{[i]}||1} = PRG(x^{i-1}_{x_{[i-1]}})$ for $i \in [1, \ldots, n-1]$
3. Output $r_{x_{[n-1]}}$

10

10a (4T) We will prove that if there exists a *PRG* $G : \{0,1\}^n \longrightarrow \{0,1\}^{n+1}$ we can construct a new *PRG* $F : \{0,1\}^n \longrightarrow \{0,1\}^{n+l}$ where l is polynomial in n.

Construction 1 : We construct the new *PRG* F as follows.

- Input:S_0 is the input of F, and $S_0 \xrightarrow{\$} \{0,1\}^n$
- $\forall j \in \{1,2,\cdots,l\}$,$(\sigma_i, S_i) := G(S_{i-1})$ where $\sigma_i \in \{0,1\}$,$S_i \in \{0,1\}^n$.
- Output : $\sigma_1\sigma_2\sigma_3\cdots\sigma_l S_l$ Theorem 2 The F described above is a PRG.

Theorem 2 The F described above is a *PRG*.

Proof. We prove this by hybrid argument. Define the hybrid H_i as follows.

- Input: $\sigma_1\sigma_2\sigma_3\cdots\sigma_l \xleftarrow{\$} \{0,1\}$,$S_i\{0,1\}^n$
- $\forall j \in \{i+1, i+2, \cdots, i+l\}$,$(\sigma_j, S_j) := G(S_{j-1})$ where $\sigma_j \in \{0,1\}$,$S_j \in \{0,1\}^n$.
- Output : $\sigma_1\sigma_2\sigma_3\cdots\sigma_l S_l$

Note that $H_0 \equiv F$ and $H_l \equiv H_{n+l}$. Assume for the purpose of contradiction that we have a non-uniform PPT adversary A that can distinguish H_0 from H_l .Define $\epsilon_i := Pr[A(H_i) = 1]$ for $i = 0, 1, \cdots, l$. Then there exists a non-negligible function v(n) such that $|\epsilon_0 - \epsilon_l| \geq v(n)$. Since $|\epsilon_0 - \epsilon_1| + |\epsilon_1 - \epsilon_2| + + \cdots + |\epsilon_{l-1} - \epsilon_l| \geq |\epsilon_0 - \epsilon_l| \geq v(n)$,

we know that there exists $k \in \{0, 1, \cdots, l-1\}$. such that $|\epsilon_k - \epsilon_{k+1}| \geq \frac{v(n)}{l}$. l is polinomial in n so $\frac{v(n)}{l}$ is also a non-negligible function. That is to say,A can distinguish H_k from H_{k+1}.Then we will construct an adversary R that can distinguish U_{n+1} from G(U_n) (which leads to a contradiction): For an input $T \in \{0,1\}^{n+1}$ T could be either from U_{n+1} or from G(U_n). Now We define $R(t) := A(H_{k+1}(\sigma_1, \sigma_2, \cdots, \sigma_k, T))$ where $\sigma_1, \sigma_2, \cdots, \sigma_k \xrightarrow{\$} \{0,1\}$ Firstly, since A and G are both PPT computable, R is also PPT computable. Further, we have

- if T is from G(U_n) then $H_{k+1}(\sigma_1, \sigma_2, \cdots, \sigma_k)$ is from H_k
- if T is from U_{n+1} then $H_{k+1}(\sigma_1, \sigma_2, \cdots, \sigma_k)$ is from H_{k+1}

Hence

$|Pr[B(G(U_n)) = 1] - Pr[B(U_{n+1}) = 1]|$
$= |Pr[A(H_k)) = 1] - Pr[A(H_{k+1})) = 1]|$
$= |\epsilon_k - \epsilon_{k+1}| \geq \frac{v(n)}{l}$,
which means R is a non-uniform PPT computable adversary to G. Contradiction to the fact that G is PRG.

(4T)

10b