

Software Security

Steffen Helke

Chair of Software Engineering

3rd December 2018



Objectives of today's lecture

- Getting to know the requirements of a strong cryptographic *pseudo-random bit generator*
- Understanding the principles of the *Pseudo One-Time-Pad*, which corresponds to the symmetric variant of $s^2\text{-mod-}n$
- Being able to apply the *asymmetric variant* of $s^2\text{-mod-}n$ based on calculating square roots of the residue class ring of n for decryption

Prime Factorization

Definition

The prime factorization of a natural number n is the product

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdot \dots \cdot p_k^{e_k}$$

where p_1, \dots, p_k are different prime numbers in pairs and the exponents are positive natural numbers, i.e. $e_1, \dots, e_k \in \mathbb{N}^+$

Algorithms

- Pollard's rho algorithm
 - Quadratic sieve algorithm
 - Number field sieve
- No polynomial algorithm for prime factor decomposition has been found yet!

Pseudo One-Time-Pad $s^2\text{-mod-}n$

Fundamentals of Number Theory

Factorization is hard

There is no polynomial algorithm to efficiently calculate the prime numbers p and q from a given n , so that $p \cdot q = n$ applies

Implications

There are two other algorithms that are as hard as factorization

- 1 Calculating a square root $\bmod n$
 - 2 Testing for a square $\bmod n$ ¹
- However, if you know p and q , then both tasks can be solved efficiently, e.g. root extraction using the CRA (Chinese Remainder Algorithm)!

¹Also called the *quadratic residuacity problem*

Two Variants for $s^2\text{-mod-}n$

Symmetric-key und Asymmetric-key Concelation

System type	Concelation		Authentication	
	sym. sym. concealation system	asym. asym. concealation system	sym. sym. authentication system	asym. digital signature system
information theoretical	Vernam-Chiffre (one-time pad)		Authentication codes	
crypto-graphically strong	Pseudo-one-time-pad with $s^2\text{-mod-}n$ -Generator			GMR
against...				
active attack				
passive attack		System with $s^2\text{-mod-}n$ -Generator		
well re-researched				
mathematical		RSA		RSA
chaos	DES/AES		DES/AES	

Source: Andreas Pfitzmann: Security in IT-Networks, 2013

Functioning of the Symmetric-key Version

Components of the secret key

- 1 Product of two prime numbers p and q with $n = p \cdot q$
- 2 Randomly chosen initial value s with $s \in \mathbb{Z}_n^*$,
where $\mathbb{Z}_n^* = \{a : \mathbb{Z}_n \mid \gcd(a, n) = 1\}$ and $\mathbb{Z}_n = \{0, \dots, n-1\}$

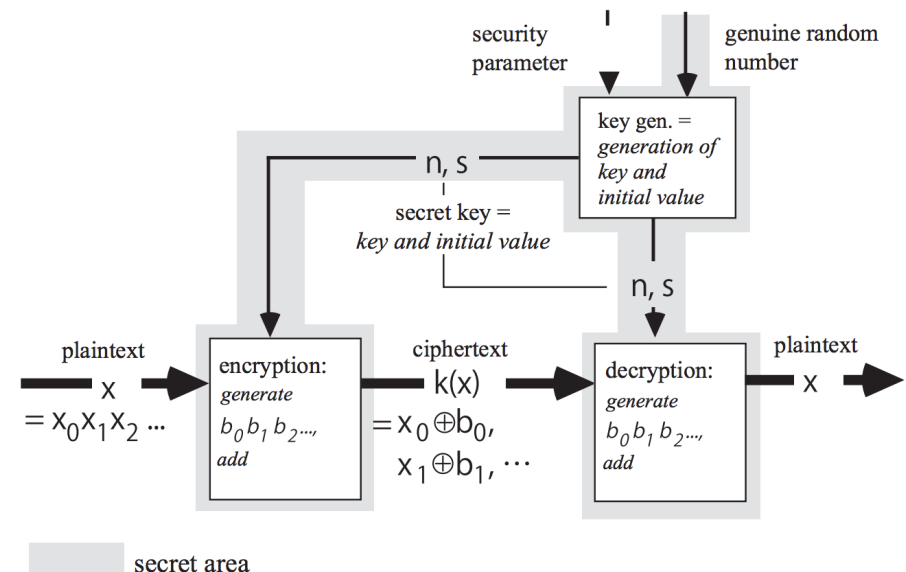
Calculation of the pseudo-random bit sequence

- 1 Calculate in each step s_{i+1} with $s_{i+1} = s_i^2 \bmod n$
- 2 Select the last bit in each step $b_i = s_i \bmod 2$

Encryption and decryption

- Add (or subtract) the pseudo-random bit sequence to the plaintext or ciphertext using the XOR-Operation similar to the One-Time Pad

Symmetric-key Version of $s^2\text{-mod-}n$



Example Calculation Symmetric-key Variant

Given the following key

- $n = 77$ with $n = 7 \cdot 11$ and initial value $s = 64$

Calculating s -sequence

- $64^2 \equiv 15 \pmod{77}$
- $15^2 \equiv 71 \pmod{77}$
- $71^2 \equiv 36 \pmod{77}$
- $36^2 \equiv 64 \pmod{77}$

Calculating Bit-sequence

- $15 \equiv 1 \pmod{2}$
- $71 \equiv 1 \pmod{2}$
- $36 \equiv 0 \pmod{2}$
- $64 \equiv 0 \pmod{2}$

Encryption

- The pseudo-random bit sequence 1100 is added to the plaintext 0011 using XOR, so that we get the ciphertext 1111

Disadvantages of Symmetric-key Encryption

Problem

Assuming n people want to communicate in pairs, so you need k different keys with $k = n \cdot (n - 1)/2$

Examples

- for $n = 100$ we obtain $k = 4.950$ keys
 - for $n = 1000$ we obtain $k = 499.500$ keys
- Quadratic increase

Solution

- Asymmetric encryption requires only $k = 2 \cdot n$ different keys
- Keys for symmetric encryption can afterwards be exchanged using asymmetric encryption

Functioning of the Asymmetric-key Version

Components Secret Key

- Two prime numbers p and q with $p \equiv q \equiv 3 \pmod{4}$

Components Public Key

- Product n with $n = p \cdot q$

Encryption

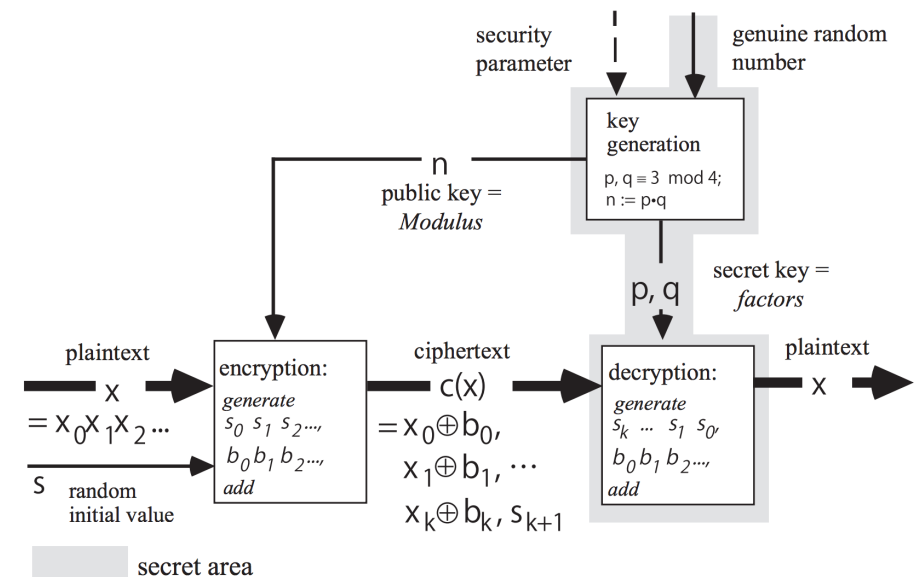
- 1 Create a bit sequence with s^2 -mod- n -method for a randomly selected start value s
- 2 Add the bit sequence to the plain text to calculate the ciphertext, then send the ciphertext and a last s_{k+1} that was not used for the encryption

Decryption

- 1 Determine the bit sequence using p and q by successively extracting the square roots² from s_{k+1}
- 2 Calculate the plaintext using XOR operation

²Only use roots that are themselves squares again!

Asymmetric-key Version of s^2 -mod- n



Computing Square Roots

Computing a square root is hard

There is no polynomial algorithm to calculate a square root for $\text{mod } n$ if you do not know the prime factors p and q

Procedure if you know the prime factors

- 1 Calculate the square roots for $\text{mod } p$ and $\text{mod } q$ using the following formulas
 - $y_p = y^{\frac{p+1}{4}} \text{ mod } p$
 - $y_q = y^{\frac{q+1}{4}} \text{ mod } q$

→ Note: Formulas are only valid for $p \equiv q \equiv 3 \text{ mod } 4$
- 2 Calculate the square root for $\text{mod } n$ from y_p and y_q using the *Chinese Remainder Algorithm* (CRA)

How to calculate the square roots for modulo p and q ?

Formulas

- $y_p = y^{\frac{p+1}{4}} \text{ mod } p$
- $y_q = y^{\frac{q+1}{4}} \text{ mod } q$

Computing the square roots

- $y_3 = 4^{\frac{3+1}{4}} = 4^1 \equiv 1 \text{ mod } 3$
- $y_7 = 4^{\frac{7+1}{4}} = 4^2 \equiv 2 \text{ mod } 7$

→ Now we have two intermediate results y_3 and y_7

Note

- The calculation rule can only be used under the condition $p \equiv q \equiv 3 \text{ mod } 4$!

Example $s^2\text{-mod-}n$ Asymmetric-key Encryption

The following prime numbers are given

- $p = 3$ and $q = 7$ with $n = p \cdot q = 21$
- We assume further on that $p \equiv q \equiv 3 \text{ mod } 4$ is fulfilled

Assumption: The following root is to be calculated

- $y = \sqrt{4} \equiv ?$

Task: How can a square root be calculated efficiently?

- $y = \sqrt{4} \equiv 2 \equiv 5 \equiv 16 \equiv 19 \text{ mod } 21$

How to combine the intermediate results with CRA?

Chinese Remainder Algorithm (CRA)

$$CRA(y_p, y_q, p, q) = u \cdot p \cdot y_q + v \cdot q \cdot y_p \text{ mod } n$$

Instantiation

$$CRA(1, 2, 3, 7) = u \cdot 3 \cdot 2 + v \cdot 7 \cdot 1 \text{ mod } 21,$$

How to calculate the base vectors u and v ?

- The integer variables u and v must fulfill the condition $\text{gcd}(3, 7) = u \cdot 3 + v \cdot 7 = 1$
- Values for u and v can be calculated using the *Extended Euclidean algorithm*

How to combine the intermediate results with CRA?

Extended Euclidean algorithm

$$\begin{aligned} 7 &= 2 \cdot 3 + 1 & (q &= s_1 \cdot p + r_1) \\ 3 &= 3 \cdot 1 + 0 & (p &= s_2 \cdot r_1 + r_2) \end{aligned}$$

In reverse order, i.e. solve all equations to the rest and then insert them step by step

$$\begin{aligned} 0 &= 3 - 3 \cdot 1 & (\text{skip this equation,} \\ & & \text{because } r_2 = 0) \\ 1 &= 1 \cdot 7 - 2 \cdot 3 & (r_1 = q - s_1 \cdot p) \end{aligned}$$

- The base vectors are $u = -2$ and $v = 1$
- Results in the square root $\text{CRA}(1, 2, 3, 7) = 16$
- **Note:** In addition, check whether 16 is a square again

Why is $s^2\text{-mod-}n$ cryptographically strong?

Question

Why is the generator $s^2\text{-mod-}n$ (symmetric variant) an unpredictable (cryptographically stronger) Pseudo-Random Bit Generator (PRBG)?

Proof obligation

Under the factorizing assumption (resp. in our proof under the quadratic-residuosity-assumption), there is no polynomial algorithm that can distinguish the random sequence generated by the PRBG from a real random sequence

Why is $s^2\text{-mod-}n$ cryptographically strong?

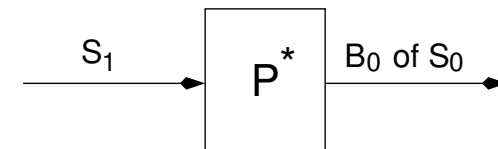
Proof by Contradiction

Assumption

There is a polynomial algorithm P which predicts the left continuation bit of a given k -bit sequence with a probability greater than $\frac{1}{2}$ (assuming that n of the residue class is also known)

Proof (First Step)

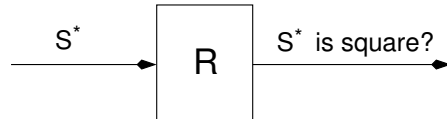
Then an algorithm P^* can be constructed from P , which calculates Bit B_0 of the initial value S_0 for a given value S_1



Continuation: Proof by Contradiction

Proof (Next Step)

Then an algorithm R can be constructed from P^* , which checks for a given value S^* with Jacoby symbol $+1$, whether S^* is a square, i.e. whether the condition $S^* \in QR_n$ holds¹



How to implement the algorithm R ?

- 1 Calculate S_1 by squaring S^* with $S_1 = (S^*)^2$
- 2 Calculate from S_1 with P^* Bit B_0 of S_0
- 3 Compare B_0 with the last bit of S^*
- 4 If the bits are identical, then $S^* = S_0$ and therefore $S^* \in QR_n$

¹ It is important that S^* has the Jacobi symbol $+1$ (see above). If you calculate the roots of a given number in the residue class ring mod n , you obtain 4 possible roots (2 with Jacobi value $+1$ & 2 with Jacobi value -1). Note, only one of the roots (Q) is a square (always Jacobi value $+1$). Further, $-Q \bmod n$ is the other root with Jacobi value $+1$. To prove the proof obligation $(S^* = S_0) \Leftrightarrow (S^* \bmod 2 = S_0 \bmod 2)$ you still need the property that n is odd, which is derivable from $p \equiv q \equiv 3 \bmod 4$

Conclusion: Proof of Contradiction

Conclusion

The algorithm R is able to perform a square test in polynomial time without knowing p and q for any S^* with Jacobi symbol $+1$, i.e. R is able to check, whether $S^* \in QR_n$

Proof (Last Step)

The derived statement is obviously in **contradiction to the quadratic residuosity assumption**, which is strongly related to the factorization assumption. This means that an attacker who can efficiently predict random numbers would also be able to factorize in polynomial time¹.

¹ In other words, if an attacker is unable to factorize efficient, he will also not be able to break the s^2 -mod- n algorithm