CSA E0 235: Cryptography 12th April, 2015

The Rabin Cryptosystem

Instructor: Arpita Patra *Submitted by: Cressida Hamlet & Marilyn George*

1 Introduction

The Rabin cryptosystem is a public key cryptosystem like Goldwasser-Micali and RSA and is more attractive than them because its security is *equivalent* to the assumption that factoring is hard. An analogous result is not known for RSA and Goldwasser-Micali base cryptosystem. So the Rabin cryptosystem is based on a weaker assumption, eventhough due to some historical factors RSA is more widely used. We will see how to compute modular squareroots and how we can use this to make a trapdoor permutation. From this trapdoor permutation we can easily formulate the Rabin cryptosystem. Throughout this section, we denote p and q for odd primes and $N = pq$.

2 Computing Modular Square Roots

The Rabin cryptosystem, receiver need to compute modular square roots. Computing square roots modulo N is easy if p and q are known, but difficult without the knowledge of P and q. We will see how to compute square roots modulo a prime and then we extend that to computing square roots modulo N.

2.1 Computing Square Roots Modulo Prime

Computing square roots modulo p is efficient, which means we can create a polynomial time algorithm to compute square roots modulo p. When we say p is a odd prime, it can be of 2 types - $p = 3\text{mod } 4$ and $p = 1\text{mod } 4$. It is easier to compute square roots when $p = 3\text{mod } 4$, but bit difficult in the other case, but for the Rabin cryptosystem we only need the case where $p = 3\text{mod } 4$. So let's see how to compute square roots modular p in this case.

$p = 3\text{mod } 4$, means there exists i such that, $p = 4i + 3$.
We know Jacobi symbol : $\mathcal{J}_p(x) = x^{\frac{p-1}{2}} \text{mod } p$.

$$\mathcal{J}_p(x) = \begin{cases} 1 & \text{if } x \in QR_p \\ -1 & \text{otherwise} \end{cases}$$

Let $a \in QR_p$, then

$a^{\frac{p-1}{2}} = 1\text{mod } p$ (multiply both sides with a)
$a^{\frac{p-1}{2}+1} = a^{\frac{p+1}{2}} = a^{2i+2} = (a^{i+1})^2 = a \text{ mod } p$
From this we get $a^{i+1} = a^{\frac{p+1}{4}}$ is a squareroot modulo p of a.
$p = 1 \text{ mod } 4$ case is shortly explained below: We will try to find an odd integer r for which $a^r = 1 \text{ mod } p$ Then as above $a^{r+1} = a \text{ mod } p$ and $a^{\frac{r+1}{2}} \text{ mod } p$ would be a square root of a since $(r + 1)/2$ is an integer. If we couldn't find such a r, we can find an element $b \in \mathbb{Z}_p^*$ and an even integer r' such that

$$a^r \cdot b^{r'} = 1 \text{ mod } p.$$

Then $a^{\frac{r+1}{2}} \cdot b^{\frac{r'}{2}} \bmod p$ is a square root of a.
Let $\frac{p-1}{2} = 2^l \cdot m$ where l, m are integers and $l \geq 1$ and m odd. Since a is a quadratic residue, we know that

$$a^{2^l m} = a^{\frac{p-1}{2}} = 1 \bmod p.$$

This means that $a^{2^{l-1}m} \bmod p$ is a square root of 1, which means ± 1. If $a^{2^{l-1}m} = 1 \bmod p$, we are in the same situation, that we can continue this till we get $a^m \bmod p = 1$ or for some $l' < l$, $a^{2^{l'}m} = -1 \bmod p$. In the first case, when $a^m \bmod p = 1$, we can get $a^{(m+1)/2} \bmod p$ as the square root. In the second case, if we have a *quadratic non-residue* $b \in \mathbb{Z}_p^*$, $b^{2^l} \bmod p$ we have

$$a^{2^{l'}m} \cdot b^{2^l m} = (-1)(-1) = +1 \bmod p.$$

With this we can proceed as before, taking the square root of the lsft-hand side to reduce the largest power of 2 dividing the exponent of a, and multiplying by $b^{2^l m}$ (whenever needed) so the right-hand side is always $+1$. Observe that the exponent of b is always divisible by a larger power of 2 than the exponent of a. We continue this till the power of a becomes odd. The algorithm to find square roots is given below.

ALGORITHM 1

Input: Prime p; quadratic residue $a \in \mathbb{Z}_p^*$
Output: A square root of a

case $p = 3 \bmod 4$:
    return $[a^{\frac{p+1}{2}} \bmod p]$
case $p = 1 \bmod 4$:
    let b be a quadratic non-residue modulo p
    compute l and m odd with $2^l \cdot m = \frac{p-1}{2}$
    $r = 2^l m$, $r' = 0$
    for $i = l$ to 1{
        $r = r/2$, $r' = r'/2$
        if $a^r \cdot b^{r'} = -1 \bmod p$
            $r' = r' + 2^l m$
    }
    return $[a^{\frac{r+1}{2}} \cdot b^{\frac{r'}{2}} \bmod p]$

2.2 Computing Square Roots Modulo N (knowing p, q)

If we know p and q, it is easy to compute squareroot modulo N. Let $a \in QR_N$ with $a \leftrightarrow (a_p, a_q)$ via Chinese remainder theorem. For a_p and a_q we can compute square roots modulo p and q respectively using the above method. If (x_p, x_q) are the square roots, then by Chinese remainder theorem, we can find x from (x_p, x_q). That is, to compute square root of a modulo an integer $N - pq$ of known factorization :

- *Compute* $a_p = [a \ mod \ p]$ *and* $a_q = [a \ mod \ q]$.
- *Using the Algorithm 1, compute a square root* x_p *of* a_p *modulo* p *and a square root* x_q *of* a_q *modulo* q.
- *Convert from the representation* $(x_p, x_q) \in \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ *to* $x \in \mathbb{Z}_N^*$ *with* $x \leftrightarrow (x_p, x_q)$. *Output* x, *which is a square root of* a *modulo* N.

It is easy to modify the algorithm so that it returns all four square roots of a.

2.3 Computing Square Roots Modulo N without knowing p and q

We have seen that computing square roots modulo N can be carried out in polynomial time if the factorization of N is known. We show here that, in contrast, computing square roots modulo a composite N of unknown factorization is as hard as factoring N.Formally, let GenModulus be a polynomial-time algorithm that, on input 1^n, outputs (N, p, q) where $N = pq$ and p and q are $n-$bit primes except with probability negligible in$n$. Consider the following experiment for a given algorithm $\mathcal{A}$ and parameter n:

The square-root computation experiment $\mathbf{SQR}_{\mathcal{A},\mathbf{GenModulus}}(n)$:

1. *Run* GenModulus(1^n) *to obtain output* N, p, q.
2. *Choose a uniform* $y \in QR_N$. *(For this we can choose a uniform* x *from* $\mathbb{Z}_N^*$ *and then square it.)*
3. $\mathcal{A}$ *is given (N,y), and outputs* $x \in \mathbb{Z}_N^*$.
4. *The output of the experiment is defined to be 1 if* $x^2 = y \bmod N$, *and 0 otherwise.*

DEFINITION 1 *We say that* computing square roots is hard relative to GenModulus *if for all probabilistic polynomial-time algorithms* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr[\mathrm{SQR}_{\mathcal{A},GenModulus}(n) = 1] \leq negl(n).$$

It is easy to see that if computing square roots is hard relative to GenModulus then factoring must be hard relative to GenModulus too: if moduli N output by GenModulus could be factored easily, then it would be easy to compute square roots modulo N by first factoring N and then applying the algorithm discussed in the previous section. Our goal now is to show the converse: that if factoring is hard relative to GenModulus then so is the problem of computing square roots. For this we will prove the following lemma.

LEMMA 1 *Let* $N = pq$ *with* p, q *distinct, odd primes. Given* $x, \hat{x}$ *such that* $x^2 = \hat{x}^2 \bmod N$ *but* $x \neq \pm\hat{x} \bmod N$, *it is possible to factor* N *in time polynomial in* $||N||$.

PROOF We claim that either $\gcd(N, x + \hat{x})$ or $\gcd(N, x - \hat{x})$ is equal to one of the prime factors of N. Since gcd computations can be carried out in polynomial time, this proves the lemma.
If $x^2 = \hat{x}^2 \bmod N$ then

$$0 = x^2 - \hat{x}^2 = (x - \hat{x}) \cdot (x + \hat{x}) \bmod N.$$

and so $N|(x - \hat{x})(x + \hat{x})$. Then $p|(x - \hat{x})(x + \hat{x})$ and so p divides one of these terms. Say $p|(x + \hat{x})$. If $q|(x + \hat{x})$ then $N|(x + \hat{x})$, but this cannot be the case since $x \neq -\hat{x} \bmod N$. So $q \nmid (x + \hat{x})$ and $\gcd(N, x + \hat{x}) = p$. Similarly we can see the $(x - \hat{x})$ case also.
So we proved the claim and so the lemma.

3 Formal reduction from the Factoring Assumption

Theorem 1. *If factoring is hard relative to* GenModulus*, then computing square roots is hard relative to* GenModulus.

Proof We now show the formal reduction of the computation of modular square roots to the standard Factoring Assumption. Recall the factoring experiment $\mathsf{Factor}_{\mathcal{A}_{\mathrm{fact}},\mathsf{GenModulus}}$ as follows:

1. Run $\mathsf{GenModulus}(1^n)$ to obtain (N, p, q).
2. $\mathcal{A}_{\text{fact}}$ is given N, and outputs $p', q' > 1$.
3. The experiment outputs 1 if $p' \cdot q' = N$, and 0 otherwise.

Note that all our assumptions are relative to a GenModulus algorithm, since an algorithm that returns composites with trivial prime factors can always be factorized. Now given our adversary $\mathcal{A}$ in the modular squareroot game, we can build $\mathcal{A}_{\text{fact}}$ as follows:

- The challenger runs GenModulus, generates (N, p, q).
- $\mathcal{A}_{\text{fact}}$ is given the modulus N as input.
- $\mathcal{A}_{\text{fact}}$ chooses a uniform $x \in \mathbb{Z}_N^*$, and computes $y := [x^2 \bmod N]$. He then runs $\mathcal{A}(N, y)$ to obtain an output $\hat{x}$.
- If $\hat{x} \neq \pm x \bmod N$, factor N as per the previous result.

From the result, we know that $\mathcal{A}_{\text{fact}}$ wins the Factor game when $\hat{x} \neq \pm x \bmod N$. Therefore:

$$\begin{aligned}\Pr[\mathsf{Factor}_{\mathcal{A}_{\text{fact}},\mathsf{GenModulus}}(n) = 1] &= \Pr[\hat{x} \neq \pm x \bmod N \wedge \hat{x}^2 = y \bmod N] \\ &= \Pr[\hat{x} \neq \pm x \bmod N | \hat{x}^2 = y \bmod N] \cdot \Pr[\hat{x}^2 = y \bmod N] \\ &= \frac{1}{2} \cdot \Pr[\mathsf{SQR}_{\mathcal{A},\mathsf{GenModulus}}(n) = 1]\end{aligned}$$

We get the last step as follows: We know from the previous Chalk & Talk on the Goldwasser-Micali cryptosystem that on picking a uniform $x \in \mathbb{Z}_N^*$, and computing $y := [x^2 \bmod N]$, we obtain a uniform y in $\mathcal{QR}_N$. Additionally N is generated by GenModulus, and hence we have that the view of $\mathcal{A}$ is identical to that of the SQR experiment. Then:

$$\Pr[\hat{x}^2 = y \bmod N] = \Pr[\mathsf{SQR}_{\mathcal{A},\mathsf{GenModulus}}(n) = 1]$$

Also, we have seen that of the 4 modular squareroots of N, two are $\pm x$ and two are $\pm\hat{x}$, then given that the SQR algorithm returns a valid squareroot, the probability $\Pr[\hat{x} \neq \pm x \bmod N | \hat{x}^2 = y \bmod N] = \frac{1}{2}$. We now complete our reduction. We have from above:

$$\Pr[\mathsf{Factor}_{\mathcal{A}_{\text{fact}},\mathsf{GenModulus}}(n) = 1] = \frac{1}{2} \cdot \Pr[\mathsf{SQR}_{\mathcal{A},\mathsf{GenModulus}}(n) = 1]$$

But we know that factoring is hard relative to GenModulus, and hence:

$$\begin{aligned}\Pr[\mathsf{Factor}_{\mathcal{A}_{\text{fact}},\mathsf{GenModulus}}(n) = 1] &\leq \mathsf{negl}(n) \\ \Pr[\mathsf{SQR}_{\mathcal{A},\mathsf{GenModulus}}(n) = 1] &\leq 2 \cdot \mathsf{negl}(n)\end{aligned}$$

Hence, this completes the proof of the theorem.

4 Rabin family of Trapdoor Permutations

From the above theorem, we can easily see the definition of a family of one-way functions based on any GenModulus relative to which factoring is hard:

- Algorithm Gen, on input 1^n, runs $\mathsf{GenModulus}(1^n)$ to get (N, p, q) and outputs $I = N$. The domain $\mathcal{D}_1$ is $\mathbb{Z}_N^*$ and the range $\mathcal{R}_1$ is $\mathcal{QR}_N$.
- Algorithm Samp, on input N, chooses a uniform element $x \in \mathbb{Z}_N^*$.
- Algorithm f, on input N and $x \in \mathbb{Z}_N^*$, outputs $[x^2 \bmod N]$.

This family is one-way if factoring is hard relative to $\mathsf{GenModulus}$.

For the purposes of a cryptosystem and efficient decryption, we can only use this if the one-way function is one-one. (i.e. every ciphertext has one decryption). We now transform this family into a family of one-way *permutations* by using moduli N of a special form, and letting $\mathcal{D}_1 \subset \mathbb{Z}_N^*$. We call $N = pq$ a *Blum integer* if p and q are distinct primes with $p = q = 3 \bmod 4$.

Proposition *Let N be a Blum integer. Then every quadratic residue modulo N has exactly one square root that is also a quadratic residue.*

Proof We see that (-1) is not a quadratic residue modulo p or q. This is because for $p = 3 \bmod 4$ it holds that $p = 4i + 3$ for some i, and we have $(-1)^{\frac{p-1}{2}} = (-1)^{2i+1} = -1 \bmod p$ (since $2i + 1$ is odd). Recall the *Jacobi symbol* notation, where $\mathcal{J}_p(x_p) = +1$, if $x_p \in \mathcal{QR}_p$ and $\mathcal{J}_p(x_p) = -1$ otherwise.

Now let $y \leftrightarrow (y_p, y_q)$ be an arbitrary quadratic residue modulo N with four square roots as $(x_p, x_q), (-x_p, x_q), (x_p, -x_q), (-x_p, -x_q)$. Exactly one of these four is $\in \mathcal{QR}_N$. Assume $\mathcal{J}_p(x_p) = +1$ and $\mathcal{J}_q(x_q) = -1$ (the proof is similar in any other case). Then

$$\mathcal{J}_q(-x_q) = \mathcal{J}_q(-1) \cdot \mathcal{J}_q(-x_q) = +1$$

Hence, $(x_p, -x_q)$ is a quadratic residue modulo N - from our earlier result that $x \leftrightarrow (x_p, x_q)$ is a quadratic residue modulo N iff $x_p \in \mathcal{QR}_p$ and $x_q \in \mathcal{QR}_q$. We also have that $\mathcal{J}_p(-x_p) = -1$, and so none of the other square roots of y are in $\mathcal{QR}_N$.

This implies that when N is a Blum integer, $f_N : \mathcal{QR}_{\mathcal{N}} \to \mathcal{QR}_N$ given by $f_N(x) = [x^2 \bmod N]$ is a *permutation* over $\mathcal{QR}_{\mathcal{N}}$. If we modify Samp above to choose a uniform element of $\mathcal{QR}_{\mathcal{N}}$, we have a family of one-way permutations. Additionally, since we can easily compute modular squareroots given the factorization of N, we have that this is a *trapdoor* family of one way permutations; often called the *Rabin family* of trapdoor permutations. Then we have:

Theorem 2. *Let $\mathsf{GenModulus}$ be an algorithm that, on input 1^n, outputs (N,p,q) where $N = pq$ and p and q are distinct primes (except maybe with negligible probability) of the form $p=q=3 \bmod 4$. If factoring is hard relative to $\mathsf{GenModulus}$, then there exists a family of trapdoor permutations.*

5 CPA-secure PKE from Trapdoor Permutations

In this section, we prove that given a family of trapdoor permutations which are known to have a hardcore predicate, we can define a CPA-secure PKE scheme *for single-bit messages.* Recalling the definition of hard-core predicate: Given a function family $\Pi = (\mathsf{Gen}, \mathsf{Samp}, f)$ - A function $\mathsf{hc} : \{0,1\}^* \to \{0,1\}$ is a *hard-core predicate* of Π if it is efficiently computable and if for every PPT algorithm $\mathcal{A}$ there is a negligible function negl such that

$$\Pr_{I \leftarrow \mathsf{Gen}(1^n), x \leftarrow \mathsf{Samp}(I)}[\mathcal{A}(I, f_I(x)) = \mathsf{hc}(I, x)] \leq \frac{1}{2} + \mathsf{negl}(n)$$

I.e. A hard-core predicate cannot be computed with probability much better than half, given only $f(x)$.

Let $\widehat{\Pi} = (\widehat{\mathsf{Gen}}, f, \mathsf{Inv})$ be a family of trapdoor permutations with hard-core predicate hc,and trapdoor td. Define a public-key cryptosystem as follows:

- Gen: On input 1^n, run $\widehat{\mathsf{Gen}}(1^n)$ to obtain (I, td). Output the public ket I, and the private key td.
- Enc: On input a public key I and a message $m \in \{0,1\}$, choose a uniform $r \in \mathcal{D}_1$ subject to the constraint that $\mathsf{hc}_I(r) = m$. Output the ciphertext $c := f_I(r)$.
- Dec: On input a private key td and a ciphertext c, compute the value $r := \mathsf{Inv}_I(c)$ and output the message $\mathsf{hc}_I(r)$.

Then we can prove that this PKE scheme is CPA-secure, by reducing to the hardness of computing the hard-core predicate $\mathsf{hc}(x)$ given only $f(x)$.

Theorem 3. *If $\widehat{\Pi}$ is a family of trapdoor permutations with hard-core predicate hc, then the above construction is CPA-secure.*

Proof Let Π denote our construction. We prove that Π has indistinguishable encryptions in the presence of an eavesdropper, since in the public-key world we know that COA Security $\Leftrightarrow$ CPA Security. We first observe that hc must be unbiased, i.e.

$$\begin{aligned} \delta_0(n) &= \Pr[\mathsf{hc}_I(x) = 0] \\ \delta_1(n) &= \Pr[\mathsf{hc}_I(x) = 1] \\ \delta_0(n), \delta_1(n) &\geq \frac{1}{2} - \mathsf{negl}(n) \end{aligned}$$

If this was not true, an attacker could simply output the more frequent bit, and break the definition of hard-core predicate. Now, let $\mathcal{A}$ be a PPT adversary. Wlog, since the scheme encrypts single-bit messages, we can assume that $m_0 = 0$ and $m_1 = 1$ in our Public Key COA experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$. We then have:

$$\begin{aligned} \Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2} \cdot \Pr[\mathcal{A}(pk, c) = 0 | c \text{ is an encryption of } 0] \\ + \frac{1}{2} \cdot \Pr[\mathcal{A}(pk, c) = 1 | c \text{ is an encryption of } 1] \end{aligned}$$

But we also have:

$$\begin{aligned} \Pr[\mathcal{A}(I, f_I(x)) = \mathsf{hc}_I] &= \delta_0(n) \cdot \Pr[\mathcal{A}(I, f_I(x)) = 0 | \mathsf{hc}_I(x) = 0] \\ &\quad + \delta_1(n) \cdot \Pr[\mathcal{A}(I, f_I(x)) = 1 | \mathsf{hc}_I(x) = 1] \\ &\geq \left(\frac{1}{2} - \mathsf{negl}(n)\right) \cdot \Pr[\mathcal{A}(I, f_I(x)) = 0 | \mathsf{hc}_I(x) = 0] \\ &\quad + \left(\frac{1}{2} - \mathsf{negl}(n)\right) \cdot \Pr[\mathcal{A}(I, f_I(x)) = 1 | \mathsf{hc}_I(x) = 1] \\ &\geq \frac{1}{2} \cdot \Pr[\mathcal{A}(I, f_I(x)) = 0 | \mathsf{hc}_I(x) = 0] \\ &\quad + \frac{1}{2} \cdot \Pr[\mathcal{A}(I, f_I(x)) = 1 | \mathsf{hc}_I(x) = 1] - 2 \cdot \mathsf{negl}(n) \\ &= \Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] - 2 \cdot \mathsf{negl}(n). \end{aligned}$$

But since hc is a hard-core predicate for $\widehat{\Pi}$, there is a negligible function negl' such that $\mathsf{negl}'(n) \geq \Pr[\mathcal{A}(I, f_I(x))$; therefore:

$$\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \mathsf{negl}'(n) - 2 \cdot \mathsf{negl}(n).$$

This completes the proof of security of the scheme.

6 Rabin Encryption Scheme

We now construct the *Rabin Encryption Scheme*, given the Rabin family of trapdoor permutations and the hard-core predicate which has been proved to be the lsb of the trapdoor permutation, similar to the case of RSA. Now in the above construction, $\mathsf{td} = (p, q)$, the factorization of $N, \widehat{\mathsf{Gen}} = \mathsf{GenModulus}, \mathcal{D}_1 = \mathcal{QR}_N$; and we have:

- Gen: On input 1^n, run $\mathsf{GenModulus}(1^n)$ to obtain (N, p, q). Output the public ket N, and the private key (p, q).
- Enc: On input a public key N and a message $m \in \{0, 1\}$, choose a uniform $x \in \mathcal{QR}_N$ subject to the constraint that $\mathsf{lsb}(x) = m$. Output the ciphertext $c := [x^2 \bmod N]$.
- Dec: On input a private key (p, q) and a ciphertext c, compute the unique $x \in \mathcal{QR}_N$ such that $x^2 := [c \bmod N]$. and output the message $\mathsf{lsb}(x)$.

Then this scheme is CPA-secure, as proved in the previous section.

Theorem 4. *If factoring is hard relative to $\mathsf{GenModulus}$, then the above construction is CPA-secure.*

7 Rabin Cryptosystem vs. RSA

- Even though the encryption schemes appear similar, the Rabin cryptosystem is *not* a special case of RSA with $e = 2$, as e is not relatively prime to $\Phi(N)$.
- In terms of security, hardness of computing modular squareroots is equivalent to hardness of factoring, whereas such a result has not been proved for the RSA Assumption, and such the Rabin scheme is based on a potentially weaker assumption.
- In terms of efficiency, the RSA and Rabin permutations are essentially the same. In fact, for large exponents e computing eth powers in RSA is slightly slower than the squaring performed as in Rabin.

However, a plain-Rabin scheme where $c = [m^2 \bmod N]$ is vulnerable to a chosen ciphertext attack, which will enable the adversary to retrieve the modular square roots and hence factor N, revealing the secret key. This is far more damaging than CC attacks on plain-RSA, which will only reveal the message, but not the secret key. This could be one of the reasons that early cryptographers preferred RSA over Rabin. In conclusion, even though RSA is more widely used than Rabin; this is more likely due to historical accident than any compelling technical justification. □

References

[1] Jonathan Katz and Yehuda Lindell, *An Introduction to Modern Cryptography*, Chapman & Hall/CRC, 2nd edition, 2014.