Primes, Factoring, and RSA
A Return to Cryptography

Foundations of Cryptography
Computer Science Department
Wellesley College

Fall 2016

Table of contents

Introduction

Generating Primes

RSA Assumption
A classic hard problem

- Given a composite integer $N$, the **factoring problem** is to find integers $p, q > 1$ such that $pq = N$.
- This problem can be solved in $O(\sqrt{N} \cdot \text{polylog}(N))$ time using **trial division**.
- Algorithms with better running time are known, but none that run in polynomial-time.

* polylog($N$) = $(\log N)^c$ for some constant $c$.
**This is not for lack of trying.

---

**Experiment in factoring**

The weak factoring experiment $w$-Factor$_A(n)$:

1. Choose two $n$-bit numbers $x_1, x_2$ at random.
2. Compute $N := x_1 \cdot x_2$.
3. $A$ is given $N$, and outputs $x'_1, x'_2 > 1$.
4. The output $N$ of the experiment is defined to be 1 if $x'_1 \cdot x'_2 = N$.

We said that the factoring problem is believed to be hard. Does this mean that for any PPT algorithm $A$ we have

$$\Pr[w$-Factor$_A(n) = 1] \leq \text{negl}(n)$$

for some negligible function negl?
Making the adversaries task harder

- The "hardest" numbers to factor seem to be those having only large prime factors.
- This suggest re-defining the above experiment so that $x_1, x_2$ are random $n$-bit primes.
- Of course, it will be necessary to generate random $n$-bit primes efficiently.

An algorithm for generating random primes

**Algorithm 8.31.**  
**Generating a random prime**

**Input:** Length $n$; parameter $t$  
**Output:** A random $n$-bit prime

\[
\text{for } i = 1 \text{ to } t : \{
    \begin{align*}
p' &\leftarrow \{0, 1\}^{n-1} \\
p &:= 1 || p' \\
    \text{if } p \text{ is prime return } p
\end{align*}
\}\text{ return fail}
\]
The distribution of primes

**Theorem 8.32** For any $n > 1$, the fraction of $n$-bit integers that are prime is at least $1/3n$.

The implication for Algorithm 8.31 is that if we set $t = 3n^2$ then the probability that a prime is not chosen in all $t$ iterations of the algorithm is at most

$$
\left(1 - \frac{1}{3n}\right)^t = \left(\left(1 - \frac{1}{3n}\right)^{3n}\right)^n \leq (e^{-1})^n = e^{-n}
$$

*Recall that for all $x \geq 1$ it holds that $(1 - 1/x)^x \leq e^{-1}$. 

Primality testing

- The problem of efficiently determining whether a given number is prime is quite old.
- It wasn’t until the 1970s that the first efficient *probabilistic* algorithms for testing primality were developed.
- A *deterministic* polynomial-time algorithm for testing primality was demonstrated in 2002, but slower in practice than the above.
Developed in the 1970s, the *Miller-Rabin* algorithm is a commonly-used. The algorithm inputs two numbers $N$ and parameter $t$. It runs in time polynomial in $\|N\|$ and $t$, and satisfies:

**Theorem 8.33.** If $N$ is prime, then the Miller-Rabin test always outputs "prime". If $N$ is composite, then the algorithm outputs "prime" with probability at most $2^{-t}$ (and outputs the correct answer "composite" with probability $1 - 2^{-t}$).

**Putting it all together**

**Algorithm 8.34. Generating a random prime**

**Input:** Length $n$; parameter $t$

**Output:** A random $n$-bit prime

```
for i = 1 to 3n^2 {
    p' <- \{0, 1\}^{n-1}
    p := 1||p'
    run the Miller-Rabin test on input p and parameter t
    if the output is "prime" return p
}
return fail
```
The factoring assumption

Let \texttt{GenModulus} be a polynomial-time algorithm that, on input $1^n$, outputs $(N, p, q)$ where $N = pq$ and $p$ are $q$ are $n$-bit primes except with negligible probability.

\textbf{The factoring experiment} $\text{Factor}_{A, \text{GenModulus}}(n)$:

1. Run \texttt{GenModulus} to obtain $(N, p, q)$.
2. $A$ is given $N$, and outputs $p', q' > 1$.
3. The output of the experiment is defined to be 1 if $p' \cdot q' = N$ and 0 otherwise.

\textit{Definition 8.45.} We say that the factoring problem is hard relative to \texttt{GenModulus} if for all PPT $A$ there exists a negligible function $\text{negl}$ such that

\[ \Pr[\text{Factor}_{A, \text{GenModulus}}(n) = 1] \leq \text{negl}(n) \]

Hard problems: The search continues

- Although the factoring assumption does yield a one-way function, in the form we have described it is not known to yield a practical cryptographic construction.

- The search goes on for other problems whose difficulty is related to the hardness of factoring.

- To date, one of the best finds were made by Rivest, Shamir, and Adleman, and is known as the RSA problem.
The RSA problem

- Recall for $N = pq$, the product of two primes, $\mathbb{Z}_N^*$ is a group of order $\phi(N) = (p - 1)(q - 1)$. If $p, q$ are known, it is easy to compute $\phi(N)$ and so computations modulo $N$ can be done by "working in the exponent modulo $\phi(N)$".
- If $p, q$ are not known, then it is difficult to compute $\phi(N)$ (in fact, computing $\phi(N)$ is as hard as factoring $N$) and "working in the exponent modulo $\phi(N)$" is not an option.
- RSA exploits this asymmetry. It is easy to solve when $\phi(N)$ is known and believed difficult otherwise.

GenRSA

Algorithm 8.47. GenRSA

*Input*: Length $n$; parameter $t$

*Output*: $N, e, d$ as described below

$(N, p, q) \leftarrow \text{GenModulus}(1^n)$

$\phi(N) := (p - 1)(q - 1)$

*find* $e$ such that $\gcd(e, \phi(N)) = 1$

*compute* $d := [e^{-1} \mod \phi(N)]$

return $N, e, d$

*N = pq* with $p, q$ $n$-bit primes.

**Such an integer $d$ exists since $e$ is invertible modulo $\phi(N)$.**
**RSA is hard relative to GenRSA**

The **RSA experiment** $\text{RSA-inv}_{A,\text{GenRSA}}(n)$:
1. Run $\text{GenRSA}(1^n)$ to obtain $(N, e, d)$.
2. Choose $y \leftarrow \mathbb{Z}_N^*$.
3. $A$ is given $N, e, y$, and outputs $x \in \mathbb{Z}_N^*$.
4. The output of the experiment is defined to be 1 if $x^e = y \mod N$, and 0 otherwise.

**Definition 8.46.** We say that the **RSA problem is hard relative to GenRSA** if for all probabilistic polynomial-time algorithms $A$ there exists a negligible function $\text{negl}$ such that

$$\Pr[\text{RSA-inv}_{A, \text{GenRSA}}(n) = 1] \leq \text{negl}(n).$$

---

**So is RSA really hard?**

- If the RSA problem is hard relative to GenRSA, then the factoring problem must be hard relative to GenModulus.
- What about the converse? In other words, suppose the factoring problem is hard relative to GenModulus, is the RSA problem hard relative to GenRSA?
- Bad news on this front; We have no proof that this is the case.