Primality Testing and Factorization

Eli Howey

MATH 56
Dartmouth College

May 27, 2014

RSA: Background

Ron Rivest, Adi Shamir, and Leonard Adleman (1977)

RSA: Background

Ron Rivest, Adi Shamir, and Leonard Adleman (1977)
Premise

RSA: Background

Ron Rivest, Adi Shamir, and Leonard Adleman (1977)
Premise

- Two keys (exponents): public and private

RSA: Background

Ron Rivest, Adi Shamir, and Leonard Adleman (1977)
Premise

- Two keys (exponents): public and private
- Requires Euler ϕ function and multiplicative inverse

RSA: Algorithm

1. Randomly choose two distinct primes p and q of approximately equal size.

RSA: Algorithm

1. Randomly choose two distinct primes p and q of approximately equal size.
2. Compute $n = pq$.

RSA: Algorithm

1. Randomly choose two distinct primes p and q of approximately equal size.
2. Compute $n = pq$.
3. Compute $\phi(n) = \phi(p)\phi(q) = (p-1)(q-1)$.

RSA: Algorithm

1. Randomly choose two distinct primes p and q of approximately equal size.
2. Compute $n = pq$.
3. Compute $\phi(n) = \phi(p)\phi(q) = (p-1)(q-1)$.
4. Choose $e \in \mathbb{Z}$ with $\gcd(e, \phi(n)) = 1$. If e is prime, then one must only check that $e \nmid \phi(n)$.

RSA: Algorithm

1. Randomly choose two distinct primes p and q of approximately equal size.
2. Compute $n = pq$.
3. Compute $\phi(n) = \phi(p)\phi(q) = (p-1)(q-1)$.
4. Choose $e \in \mathbb{Z}$ with $\gcd(e, \phi(n)) = 1$. If e is prime, then one must only check that $e \nmid \phi(n)$.
5. Find the multiplicative inverse $d = e^{-1} \mod \phi(n)$.

RSA: Algorithm

1. Randomly choose two distinct primes p and q of approximately equal size.
2. Compute $n = pq$.
3. Compute $\phi(n) = \phi(p)\phi(q) = (p-1)(q-1)$.
4. Choose $e \in \mathbb{Z}$ with $\gcd(e, \phi(n)) = 1$. If e is prime, then one must only check that $e \nmid \phi(n)$.
5. Find the multiplicative inverse $d = e^{-1} \mod \phi(n)$.
6. The public key is defined as the pair (e, n), and the private key as (d, n).

RSA: Encryption/Decryption

Given a message M, $0 \leq M < n$ (or ciphertext C):

RSA: Encryption/Decryption

Given a message M, $0 \leq M < n$ (or ciphertext C):

- Encryption: $E(M) \equiv M^e \mod n$ (only requires public key)

RSA: Encryption/Decryption

Given a message M, $0 \leq M < n$ (or ciphertext C):

- Encryption: $E(M) \equiv M^e \mod n$ (only requires public key)
- Decryption: $D(C) \equiv M^d \mod n$ (requires private key)

RSA: Encryption/Decryption

Given a message M, $0 \leq M < n$ (or ciphertext C):

- Encryption: $E(M) \equiv M^e \mod n$ (only requires public key)
- Decryption: $D(C) \equiv M^d \mod n$ (requires private key)

Private key functionally impossible to crack without p, q

Primality Testing

Two major (deterministic) methods:

Primality Testing

Two major (deterministic) methods:

- Trial division

Primality Testing

Two major (deterministic) methods:

- Trial division
  - $O(n^{1/2} \ln n \, (\ln \ln n)^2)$

Primality Testing

Two major (deterministic) methods:

- Trial division
  - $O(n^{1/2} \ln n \, (\ln \ln n)^2)$
  - Best for $n < 10^{10}$

Primality Testing

Two major (deterministic) methods:

- Trial division
  - $O(n^{1/2} \ln n \, (\ln \ln n)^2)$
  - Best for $n < 10^{10}$
- Sieve of Eratosthenes

Primality Testing

Two major (deterministic) methods:

- Trial division
  - $O(n^{1/2} \ln n \, (\ln \ln n)^2)$
  - Best for $n < 10^{10}$
- Sieve of Eratosthenes
  - $O(\ln \ln n)$ per sieve element

Primality Testing

Two major (deterministic) methods:

- Trial division
  - $O(n^{1/2} \ln n \, (\ln \ln n)^2)$
  - Best for $n < 10^{10}$
- Sieve of Eratosthenes
  - $O(\ln \ln n)$ per sieve element
  - Good for $n < 10^{12}$, but segmenting can extend range

Factorization Methods

Covers four method classes

Factorization Methods

Covers four method classes

- Methods covered in class
  - Trial division (sieve implementation)
  - Fermat's method

Factorization Methods

Covers four method classes

- Methods covered in class
  - Trial division (sieve implementation)
  - Fermat's method
- Probabilistic: Pollard ρ method

Factorization Methods

Covers four method classes

- Methods covered in class
  - Trial division (sieve implementation)
  - Fermat's method
- Probabilistic: Pollard ρ method
- Quadratic sieve (QS) & number field sieve (NFS)

Factorization Methods

Covers four method classes

- Methods covered in class
  - Trial division (sieve implementation)
  - Fermat's method
- Probabilistic: Pollard ρ method
- Quadratic sieve (QS) & number field sieve (NFS)
- Factoring multiple RSA keys with Batch GCD

Pollard ρ method: Premise

Let $n \in \mathbb{Z}^+$ be composite with least prime factor p.
$\mathcal{S} = \{1, 2, \ldots, p-1\}$ $\qquad$ $f(x) = x^2 + a \mod p, a \in \mathcal{S}$
For all $s \in \mathcal{S}$, the sequence

$$s, f(s), f(f(s)), \ldots$$

eventually becomes cyclic (after $O(\sqrt{p})$ iterations).

$$F(x) = x^2 + a \mod n \equiv f(x) \mod p$$

[Floyd] $\exists i$ such that $2i = O(\sqrt{p})$ and $F^{(i)}(s) \equiv F^{(2i)}(s) \mod p$
$\implies \gcd(F^{(i)}(s) - F^{(2i)}(s), n)$ is a factor (if $\neq n$)

Pollard ρ method: Algorithm

1. Randomly choose integers $a \in [1, n-3]$ and $s \in [0, n-1]$.

Pollard ρ method: Algorithm

1. Randomly choose integers $a \in [1, n-3]$ and $s \in [0, n-1]$.
2. Define $F(x) = (x^2 + a) \mod n$.

Pollard ρ method: Algorithm

1. Randomly choose integers $a \in [1, n-3]$ and $s \in [0, n-1]$.
2. Define $F(x) = (x^2 + a) \mod n$.
3. Set $U = V = s$.

Pollard ρ method: Algorithm

1. Randomly choose integers $a \in [1, n-3]$ and $s \in [0, n-1]$.
2. Define $F(x) = (x^2 + a) \mod n$.
3. Set $U = V = s$.
4. Iterate U and V as follows:
   - $U = F(U)$,
   - $V = F^{(2)}(V)$.

Pollard ρ method: Algorithm

1. Randomly choose integers $a \in [1, n-3]$ and $s \in [0, n-1]$.
2. Define $F(x) = (x^2 + a) \mod n$.
3. Set $U = V = s$.
4. Iterate U and V as follows:
   - $U = F(U)$,
   - $V = F^{(2)}(V)$.
5. Calculate $g = \gcd(U - V, n)$.

Pollard ρ method: Algorithm

1. Randomly choose integers $a \in [1, n-3]$ and $s \in [0, n-1]$.
2. Define $F(x) = (x^2 + a) \mod n$.
3. Set $U = V = s$.
4. Iterate U and V as follows:
   - $U = F(U)$,
   - $V = F^{(2)}(V)$.
5. Calculate $g = \gcd(U - V, n)$.
6. If $g = 1$, go back to Step 4. If $g = n$, go back to Step 1.

Pollard ρ method: Algorithm

1. Randomly choose integers $a \in [1, n-3]$ and $s \in [0, n-1]$.
2. Define $F(x) = (x^2 + a) \mod n$.
3. Set $U = V = s$.
4. Iterate U and V as follows:
   - $U = F(U)$,
   - $V = F^{(2)}(V)$.
5. Calculate $g = \gcd(U - V, n)$.
6. If $g = 1$, go back to Step 4. If $g = n$, go back to Step 1.
7. Return g.

Pollard ρ method: Runtime

- Step 4 iteration occurs $\leq \frac{1}{2} O(\sqrt{p})$ times
- $O(\ln n \ln \ln n)$ per iteration

Pollard ρ method: Runtime

- Step 4 iteration occurs $\leq \frac{1}{2} O(\sqrt{p})$ times
- $O(\ln n \ln \ln n)$ per iteration

Total runtime:

$$O(p^{1/2} \ln n \ln \ln n) \approx O(n^{1/4} \ln n \ln \ln n)$$

Pollard ρ method: Runtime

- Step 4 iteration occurs $\leq \frac{1}{2} O(\sqrt{p})$ times
- $O(\ln n \ln \ln n)$ per iteration

Total runtime:

$$O(p^{1/2} \ln n \ln \ln n) \approx O(n^{1/4} \ln n \ln \ln n)$$

Probabilistic (choice of a and s affects result)

Quadratic Sieve (QS)

Establish quadratic congruence among products of B-smooth candidates

$$x^2 \equiv y^2 \mod n$$

Quadratic Sieve (QS)

Establish quadratic congruence among products of B-smooth candidates

$$x^2 \equiv y^2 \mod n$$

Optimal B [Pomerance]:

$$B \approx \exp(\frac{1}{2}\sqrt{\ln n \ln \ln n})$$

Quadratic Sieve (QS)

Establish quadratic congruence among products of B-smooth candidates

$$x^2 \equiv y^2 \mod n$$

Optimal B [Pomerance]:

$$B \approx \exp(\frac{1}{2}\sqrt{\ln n \ln \ln n})$$

Runtime:

$$O(B^2) \approx L(n) = O(\exp(\sqrt{\ln n \ln \ln n}))$$

NFS: Number Fields

Let f be a polynomial with coeffs. in $\mathbb{Z}$, degree k, root $r \in \mathbb{C}$. f is *irreducible* if f cannot be expressed as the product of two polynomials with coeffs. in $\mathbb{Z}$, degree $< k$. If f is irreducible, we can define the *polynomial ring* (number field)

$$\mathbb{Z}[r] = \{c_{k-1}r^{k-1} + \cdots + c_1 r + c_0 \in \mathbb{R} \mid c_0, c_1, \ldots, c_{k-1} \in \mathbb{Z}\}$$

Multiplication in $\mathbb{Z}[r]$: polynomial multiplication, then reduction mod r^k

NFS: Concept

Establish quadratic congruence among products of B-smooth products of polynomials in $\mathbb{Z}[x]$

$$u^2 \equiv v^2 \mod n,$$

where v^2 is the product of $a - bm$ for pairs (a, b) such that $F(a, b)G(a, b)$ is B-smooth, and m an approximate root of n

NFS: Algorithm

Setup:

- Set $d = \lfloor (3 \ln n / \ln \ln n) \rfloor$.
- Set $B = \lfloor \exp((8/9)^{1/3} (\ln n)^{1/3} (\ln \ln n)^{2/3}) \rfloor$.[1]
- Set $m = \lfloor n^{1/d} \rfloor$.
- Write n in base m: $n = m^d + c_{d-1} m^{d-1} + \cdots + c_0$.
- Define $f(x) = x^d + c_{d-1} x^{d-1} + \cdots + c_0$. Note $f(m) = n$.
- Attempt to factor f into irreducible polynomials $g, h \in \mathbb{Z}[x]$ and, if it factors, return $n = g(m)h(m)$.
- Define $F(x, y) = x^d + c_{d-1} x^{d-1} y + \cdots + c_0 y^d$.
- Define $G(x, y) = x - my$.

[1] The values of d and B can be tuned to taste; these are experimentally determined optimal values [Pomerance].

NFS: Algorithm

Setup (continued):

- Compute $R(p) = \{r \in [0, p-1] \mid f(r) \equiv 0 \mod p\}$ for each prime $p \leq B$.
- Set $k = \lfloor 3 \lg n \rfloor$.
- Set $B' = \sum_{p \leq B} \#R(p)$.
- Set $V = 1 + \pi(B) + B' + k$.
- Set $M = B$.

NFS: Algorithm

1. Sieve for a set $\mathcal{S}'$ of (at least) $V + 1$ coprime integer pairs (a, b) with $0 < |a|, b \leq M$, and $F(a, b)G(a, b)$ B-smooth. If this fails, increase M and retry this step.

NFS: Algorithm

1. Sieve for a set $\mathcal{S}'$ of (at least) $V+1$ coprime integer pairs (a,b) with $0 < |a|, b \leq M$, and $F(a,b)G(a,b)$ B-smooth. If this fails, increase M and retry this step.
2. Create an exponent matrix from the pairs from Step 2. Each row will be the exponent vector for $a - b\alpha$ for some root α of f.

NFS: Algorithm

1. Sieve for a set $\mathcal{S}'$ of (at least) $V + 1$ coprime integer pairs (a, b) with $0 < |a|, b \leq M$, and $F(a, b)G(a, b)$ B-smooth. If this fails, increase M and retry this step.
2. Create an exponent matrix from the pairs from Step 2. Each row will be the exponent vector for $a - b\alpha$ for some root α of f.
3. Use linear algebra to find a subset $\mathcal{S}$ of $\mathcal{S}'$ whose elementwise sum is the zero vector (e.g., Block Lanczos).

NFS: Algorithm

1. Sieve for a set $\mathcal{S}'$ of (at least) $V+1$ coprime integer pairs (a,b) with $0 < |a|, b \leq M$, and $F(a,b)G(a,b)$ B-smooth. If this fails, increase M and retry this step.
2. Create an exponent matrix from the pairs from Step 2. Each row will be the exponent vector for $a - b\alpha$ for some root α of f.
3. Use linear algebra to find a subset $\mathcal{S}$ of $\mathcal{S}'$ whose elementwise sum is the zero vector (e.g., Block Lanczos).
4. Compute v such that $\prod_{(a,b)\in\mathcal{S}}(a - bm) \equiv v^2 \mod n$.

NFS: Algorithm

1. Sieve for a set $\mathcal{S}'$ of (at least) $V+1$ coprime integer pairs (a,b) with $0 < |a|, b \leq M$, and $F(a,b)G(a,b)$ B-smooth. If this fails, increase M and retry this step.
2. Create an exponent matrix from the pairs from Step 2. Each row will be the exponent vector for $a - b\alpha$ for some root α of f.
3. Use linear algebra to find a subset $\mathcal{S}$ of $\mathcal{S}'$ whose elementwise sum is the zero vector (e.g., Block Lanczos).
4. Compute v such that $\prod_{(a,b)\in\mathcal{S}}(a-bm) \equiv v^2 \mod n$.
5. Find a γ in $\mathbb{Z}[\alpha]$ such that $\left(f'(m)^2 \prod_{(a,b)\in\mathcal{S}}(a-bm)\right) \equiv \gamma^2 \mod n$.

NFS: Algorithm

1. Sieve for a set $\mathcal{S}'$ of (at least) $V+1$ coprime integer pairs (a,b) with $0 < |a|, b \leq M$, and $F(a,b)G(a,b)$ B-smooth. If this fails, increase M and retry this step.
2. Create an exponent matrix from the pairs from Step 2. Each row will be the exponent vector for $a - b\alpha$ for some root α of f.
3. Use linear algebra to find a subset $\mathcal{S}$ of $\mathcal{S}'$ whose elementwise sum is the zero vector (e.g., Block Lanczos).
4. Compute v such that $\prod_{(a,b)\in\mathcal{S}}(a-bm) \equiv v^2 \mod n$.
5. Find a γ in $\mathbb{Z}[\alpha]$ such that $\left(f'(m)^2 \prod_{(a,b)\in\mathcal{S}}(a-bm)\right) \equiv \gamma^2 \mod n$.
6. Compute $u \equiv \phi(\gamma) \mod n$, where $\phi : \mathbb{Z}[x] \to \mathbb{Z}/n\mathbb{Z}$ a homomorphism.

NFS: Algorithm

1. Sieve for a set $\mathcal{S}'$ of (at least) $V+1$ coprime integer pairs (a,b) with $0 < |a|, b \leq M$, and $F(a,b)G(a,b)$ B-smooth. If this fails, increase M and retry this step.
2. Create an exponent matrix from the pairs from Step 2. Each row will be the exponent vector for $a - b\alpha$ for some root α of f.
3. Use linear algebra to find a subset $\mathcal{S}$ of $\mathcal{S}'$ whose elementwise sum is the zero vector (e.g., Block Lanczos).
4. Compute v such that $\prod_{(a,b)\in\mathcal{S}}(a-bm) \equiv v^2 \mod n$.
5. Find a γ in $\mathbb{Z}[\alpha]$ such that $\left(f'(m)^2 \prod_{(a,b)\in\mathcal{S}}(a-bm)\right) \equiv \gamma^2 \mod n$.
6. Compute $u \equiv \phi(\gamma) \mod n$, where $\phi : \mathbb{Z}[x] \to \mathbb{Z}/n\mathbb{Z}$ a homomorphism.
7. Return $\gcd(u - f'(m)v, n)$.

NFS: Runtime

Complexity given heuristically [Pomerance], proof eludes us!

$$O\left(\exp\left(\sqrt[3]{\frac{64}{9}}+o(1)\right)(\ln n)^{1/3}(\ln\ln n)^{2/3}\right)$$

Compare to quadratic sieve:

$$O(\exp(\sqrt{\ln n \ln\ln n}))$$

Batch GCD

- RSA keys may be created using faulty prime number generators

Batch GCD

- RSA keys may be created using faulty prime number generators
- If two keys share a factor, then we can factor both keys!

Batch GCD

- RSA keys may be created using faulty prime number generators
- If two keys share a factor, then we can factor both keys!

Given keys $\{N_1, N_2, \ldots, N_m\}$, calculate

$$\gcd(N_1, N_1 N_2 \cdots N_m),$$

$$\gcd(N_2, N_1 N_3 \cdots N_m),$$

$$\ldots$$

$$\gcd(N_m, N_1 N_2 \cdots N_{m-1})$$

Batch GCD: Algorithm

Given a list $\{N_1, N_2, \ldots, N_m\}$ of RSA keys:

Batch GCD: Algorithm

Given a list $\{N_1, N_2, \ldots, N_m\}$ of RSA keys:

1. Calculate $N = N_1 N_2 \cdots N_m$.

Batch GCD: Algorithm

Given a list $\{N_1, N_2, \ldots, N_m\}$ of RSA keys:

1. Calculate $N = N_1 N_2 \cdots N_m$.
2. Create a list G of length m.

Batch GCD: Algorithm

Given a list $\{N_1, N_2, \ldots, N_m\}$ of RSA keys:

1. Calculate $N = N_1 N_2 \cdots N_m$.
2. Create a list G of length m.
3. For each $i = 1, 2, \ldots, m$:
   - Calculate $R_i \equiv N \mod N_i^2$.

Batch GCD: Algorithm

Given a list $\{N_1, N_2, \ldots, N_m\}$ of RSA keys:

1. Calculate $N = N_1 N_2 \cdots N_m$.
2. Create a list G of length m.
3. For each $i = 1, 2, \ldots, m$:
   - Calculate $R_i \equiv N \mod N_i^2$.
   - Set $G_i = \gcd(N_i, R_i/N_i) = \gcd(N_i, N/N_i)$.

Batch GCD: Algorithm

Given a list $\{N_1, N_2, \ldots, N_m\}$ of RSA keys:

1. Calculate $N = N_1 N_2 \cdots N_m$.
2. Create a list G of length m.
3. For each $i = 1, 2, \ldots, m$:
   - Calculate $R_i \equiv N \mod N_i^2$.
   - Set $G_i = \gcd(N_i, R_i/N_i) = \gcd(N_i, N/N_i)$.
4. Return G.

Batch GCD: Considerations

$\{N_i \mid G_i > 1\}$ is the set of keys that share a factor with some other key

1. G_i prime $\implies$ G_i nontrivial factor of N_i
2. $G_i = 1$ or composite $\implies$ find pairwise GCDs until (1)

Complexity:

$$O(m\,(\ln n)^2)$$

For RSA-specific cracks, may be much more effective than any other method listed

Used to crack thousands of RSA keys from a set of size $O(10^7)$ (2011)

Future of Factorization

Ever-growing push for better methods
Current research