

Factoring and Discrete Log

Nadia Heninger

University of Pennsylvania

June 1, 2015



A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

R.L. Rivest, A. Shamir, and L. Adleman*

Textbook RSA

[Rivest Shamir Adleman 1977]

Public Key

N = pq modulus

e encryption exponent

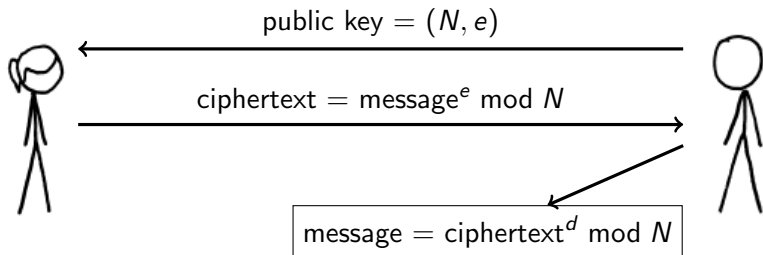
Private Key

p, q primes

d decryption exponent

$$(d = e^{-1} \bmod (p-1)(q-1))$$

Encryption



Textbook RSA

[Rivest Shamir Adleman 1977]

Public Key

N = pq modulus

e encryption exponent

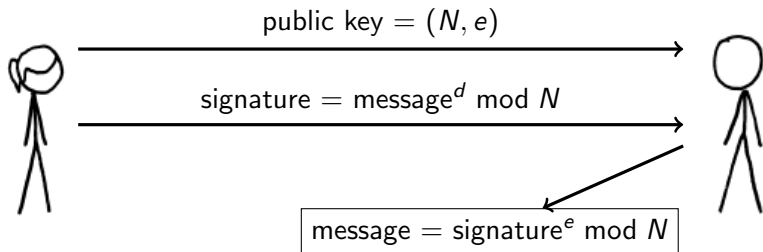
Private Key

p, q primes

d decryption exponent

$$(d = e^{-1} \bmod (p-1)(q-1))$$

Signing



Computational problems

Factoring

Problem: Given N , compute its prime factors.

- ▶ Computationally equivalent to computing private key d .
- ▶ Factoring is in NP and coNP \rightarrow not NP-complete (unless $P=NP$ or similar).

Computational problems

eth roots mod N

Problem: Given N , e , and c , compute x such that $x^e \equiv c \pmod{N}$.

- ▶ Equivalent to decrypting an RSA-encrypted ciphertext.
- ▶ Equivalent to selective forgery of RSA signatures.
- ▶ Conflicting results about whether it reduces to factoring:
 - ▶ “Breaking RSA may not be equivalent to factoring” [Boneh Venkatesan 1998]
“an algebraic reduction from factoring to breaking low-exponent RSA can be converted into an efficient factoring algorithm”
 - ▶ “Breaking RSA generically is equivalent to factoring” [Aggarwal Maurer 2009]
“a generic ring algorithm for breaking RSA in \mathbb{Z}_N can be converted into an algorithm for factoring”
- ▶ “RSA assumption”: This problem is hard.

A garden of attacks on textbook RSA

Unpadded RSA encryption is homomorphic under multiplication.
Let's have some fun!

Attack: Malleability

Given a ciphertext $c = \text{Enc}(m) = m^e \bmod N$, attacker can forge ciphertext $\text{Enc}(ma) = ca^e \bmod N$ for any a .

Attack: Chosen ciphertext attack

Given a ciphertext $c = \text{Enc}(m)$ for unknown m , attacker asks for $\text{Dec}(ca^e \bmod N) = d$ and computes $m = da^{-1} \bmod N$.

Attack: Signature forgery

Attacker wants $\text{Sign}(x)$. Attacker computes $z = xy^e \bmod N$ for some y and asks signer for $s = \text{Sign}(z) = z^d \bmod N$. Attacker computes $\text{Sign}(z) = sy^{-1} \bmod N$.

So in practice **always use padding on messages**.

A CRYPTO NERD'S IMAGINATION:

HIS LAPTOP'S ENCRYPTED.
LET'S BUILD A MILLION-DOLLAR
CLUSTER TO CRACK IT.

BLAST! OUR
EVIL PLAN
IS FOILED!

NO GOOD! IT'S
4096-BIT RSA!



WHAT WOULD ACTUALLY HAPPEN:

HIS LAPTOP'S ENCRYPTED.
DRUG HIM AND HIT HIM WITH
THIS \$5 WRENCH UNTIL
HE TELLS US THE PASSWORD.

GOT IT.



So how hard *is* factoring?

So how hard *is* factoring?

```
sage: time factor(random_prime(2^32)*random_prime(2^32))
```

So how hard *is* factoring?

```
sage: time factor(random_prime(2^32)*random_prime(2^32))
CPU times: user 3.96 ms, sys: 95 s, total: 4.06 ms
Wall time: 4 ms
1162180681 * 3036055123
```

So how hard *is* factoring?

```
sage: time factor(random_prime(2^32)*random_prime(2^32))
```

```
CPU times: user 3.96 ms, sys: 95 s, total: 4.06 ms
```

```
Wall time: 4 ms
```

```
1162180681 * 3036055123
```

```
sage: time factor(random_prime(2^64)*random_prime(2^64))
```

So how hard *is* factoring?

```
sage: time factor(random_prime(2^32)*random_prime(2^32))
CPU times: user 3.96 ms, sys: 95 s, total: 4.06 ms
Wall time: 4 ms
1162180681 * 3036055123
```

```
sage: time factor(random_prime(2^64)*random_prime(2^64))
CPU times: user 65.1 ms, sys: 15.4 ms, total: 80.4 ms
Wall time: 136 ms
3467882422082372663 * 9260649369177772927
```

So how hard *is* factoring?

```
sage: time factor(random_prime(2^32)*random_prime(2^32))
CPU times: user 3.96 ms, sys: 95 s, total: 4.06 ms
Wall time: 4 ms
1162180681 * 3036055123
```

```
sage: time factor(random_prime(2^64)*random_prime(2^64))
CPU times: user 65.1 ms, sys: 15.4 ms, total: 80.4 ms
Wall time: 136 ms
3467882422082372663 * 9260649369177772927
```

```
sage: time factor(random_prime(2^96)*random_prime(2^96))
```

So how hard *is* factoring?

```
sage: time factor(random_prime(2^32)*random_prime(2^32))
CPU times: user 3.96 ms, sys: 95 s, total: 4.06 ms
Wall time: 4 ms
1162180681 * 3036055123
```

```
sage: time factor(random_prime(2^64)*random_prime(2^64))
CPU times: user 65.1 ms, sys: 15.4 ms, total: 80.4 ms
Wall time: 136 ms
3467882422082372663 * 9260649369177772927
```

```
sage: time factor(random_prime(2^96)*random_prime(2^96))
CPU times: user 5.48 s, sys: 92.7 ms, total: 5.57 s
Wall time: 5.74 s
35446974246595767622419590689 * 62426036507249326299176493091
```

So how hard *is* factoring?

```
sage: time factor(random_prime(2^32)*random_prime(2^32))
CPU times: user 3.96 ms, sys: 95 s, total: 4.06 ms
Wall time: 4 ms
1162180681 * 3036055123
```

```
sage: time factor(random_prime(2^64)*random_prime(2^64))
CPU times: user 65.1 ms, sys: 15.4 ms, total: 80.4 ms
Wall time: 136 ms
3467882422082372663 * 9260649369177772927
```

```
sage: time factor(random_prime(2^96)*random_prime(2^96))
CPU times: user 5.48 s, sys: 92.7 ms, total: 5.57 s
Wall time: 5.74 s
35446974246595767622419590689 * 62426036507249326299176493091
```

```
sage: time factor(random_prime(2^128)*random_prime(2^128))
```


So how hard *is* factoring?

```
sage: time factor(random_prime(2^32)*random_prime(2^32))
CPU times: user 3.96 ms, sys: 95 s, total: 4.06 ms
Wall time: 4 ms
1162180681 * 3036055123
```

```
sage: time factor(random_prime(2^64)*random_prime(2^64))
CPU times: user 65.1 ms, sys: 15.4 ms, total: 80.4 ms
Wall time: 136 ms
3467882422082372663 * 9260649369177772927
```

```
sage: time factor(random_prime(2^96)*random_prime(2^96))
CPU times: user 5.48 s, sys: 92.7 ms, total: 5.57 s
Wall time: 5.74 s
35446974246595767622419590689 * 62426036507249326299176493091
```

```
sage: time factor(random_prime(2^128)*random_prime(2^128))
CPU times: user 12min 9s, sys: 5.63 s, total: 12min 15s
Wall time: 12min 24s
199096156382647146999656258302432743511 * 3104632787128844624746
```

Factoring in practice

Two families of factoring algorithms:

1. Algorithms whose running time depends on the size of the factor to be found.
 - ▶ Good for factoring small numbers, and finding small factors of big numbers.
2. Algorithms whose running time depends on the size of the number to be factored.
 - ▶ Good for factoring big numbers with big factors.

Trial Division

Good for finding very small factors

Takes $p/\log p$ trial divisions to find a prime factor p .

Pollard rho

Good for finding slightly larger prime factors

Intuition

- ▶ Try to take a random walk among elements mod N .
- ▶ If p divides N , there will be a cycle of length p .
- ▶ Expect a collision after searching about \sqrt{p} random elements.

Pollard rho

Good for finding slightly larger prime factors

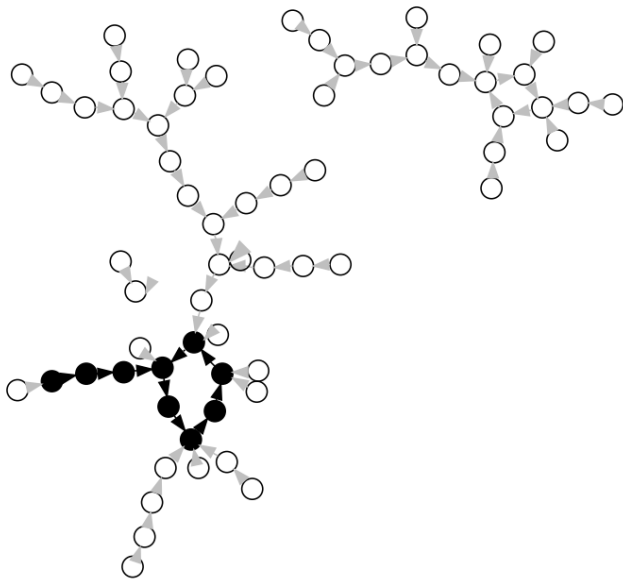
Intuition

- ▶ Try to take a random walk among elements mod N .
- ▶ If p divides N , there will be a cycle of length p .
- ▶ Expect a collision after searching about \sqrt{p} random elements.

Details

- ▶ “Random” function: $f(x) = x^2 + c \bmod N$ for random c .
- ▶ For random starting point a , compute $a, f(a), f(f(a)), \dots$
- ▶ Naive implementation uses \sqrt{p} memory, $O(1)$ lookup time.
- ▶ To reduce memory:
 - ▶ Floyd cycle-finding algorithm: Store two pointers, and move one twice as fast as the other until they coincide.
 - ▶ Method of distinguished points: Store points satisfying easily tested property like k leading zeros.

Why is it called the rho algorithm?



Pollard rho in Sage

```
def rho(n):  
    a = 98357389475943875; c=10 # some random values  
    f = lambda x: (x^2+c)%n  
  
    a1 = f(a) ; a2 = f(a1)  
    while gcd(n, a2-a1)==1:  
        a1 = f(a1); a2 = f(f(a2))  
    return gcd(n, a2-a1)  
  
sage: N = 698599699288686665490308069057420138223871  
sage: rho(N)  
2053
```

Reminders: Orders and groups

Theorem (Fermat's Little Theorem)

$a^{p-1} \equiv 1 \pmod p$ for any $0 < a < p$.

Let $\text{ord}(a)_p$ be the order of $a \pmod p$. (Smallest positive integer such that $a^{\text{ord}(a)_p} \equiv 1 \pmod p$.)

Theorem (Lagrange)

$\text{ord}(a)_p$ divides $p - 1$.

Pollard's $p - 1$ method

Good for finding special small factors

Intuition

- ▶ If $a^r \equiv 1 \pmod{p}$ then $r \mid \text{ord}(a)_p$ and $p \mid \gcd(a^r - 1, N)$.
- ▶ Don't know p , pick very smooth number r , hoping for $\text{ord}(a)_p$ to divide it.

Definition: An integer is B -smooth if all its prime factors are $\leq B$.

Pollard's $p - 1$ method

Good for finding special small factors

Intuition

- ▶ If $a^r \equiv 1 \pmod{p}$ then $r \mid \text{ord}(a)_p$ and $p \mid \gcd(a^r - 1, N)$.
- ▶ Don't know p , pick very smooth number r , hoping for $\text{ord}(a)_p$ to divide it.

Definition: An integer is B -smooth if all its prime factors are $\leq B$.

```
N=44426601460658291157725536008128017297890787
4637194279031281180366057
r=lcm(range(1,2^22)) # this takes a while ...
s=Integer(pow(2,r,N))
sage: gcd(s-1,N)
1267650600228229401496703217601
```

Pollard $p - 1$ method

- ▶ This method finds larger factors than the rho method (in the same time)
...but only works for special primes.

In the previous example,

$$p - 1 = 2^6 \cdot 3^2 \cdot 5^2 \cdot 17 \cdot 227 \cdot 491 \cdot 991 \cdot 36559 \cdot 308129 \cdot 4161791$$

has only small factors (aka. $p - 1$ is *smooth*).

- ▶ Many crypto standards require using only “*safe primes*” a.k.a primes where $p - 1 = 2q + 1$, so $p - 1$ is really non-smooth.
- ▶ This recommendation is outdated for RSA. The elliptic curve method (next slide) works even for “safe” primes.

Lenstra's Elliptic Curve Method

Good for finding medium-sized factors

Intuition

- ▶ Pollard's $p - 1$ method works in the multiplicative group of integers modulo p .
- ▶ The elliptic curve method is exactly the $p - 1$ method, but over the group of points on an elliptic curve modulo p :
 - ▶ Multiplication of group elements becomes addition of points on the curve.
 - ▶ All arithmetic is still done modulo N .
 - ▶ Tanja will tell you much more about elliptic curves.

Lenstra's Elliptic Curve Method

Good for finding medium-sized factors

Intuition

- ▶ Pollard's $p - 1$ method works in the multiplicative group of integers modulo p .
- ▶ The elliptic curve method is exactly the $p - 1$ method, but over the group of points on an elliptic curve modulo p :
 - ▶ Multiplication of group elements becomes addition of points on the curve.
 - ▶ All arithmetic is still done modulo N .
 - ▶ Tanja will tell you much more about elliptic curves.

Theorem (Hasse)

The order of an elliptic curve modulo p is in $[p + 1 - 2\sqrt{p}, p + 1 + 2\sqrt{p}]$.

There are lots of smooth numbers in this interval.

If one elliptic curve doesn't work, try until you find a smooth order.

Elliptic Curves in Sage

```
def curve(d):
    frac_n = type(d)
    class P(object):
        def __init__(self,x,y):
            self.x,self.y = frac_n(x),frac_n(y)

        def __add__(a,b):
            return P((a.x*b.y + b.x*a.y)/(1 + d*a.x*a.y*b.x*b.y)
                    (a.y*b.y - a.x*b.x)/(1 - d*a.x*b.x*a.y*b.y))

        def __mul__(self, m):
            return double_and_add(self,m,P(0,1))

    ...
```

Elliptic Curve Factorization

```
def ecm(n,y,t):  
    # Choose a curve and a point on the curve.  
    frac_n = Q(n)  
    P = curve(frac_n(1,3))  
    p = P(2,3)  
  
    q = p * lcm(xrange(1,y))  
    return gcd(q.x.t,n)
```

- ▶ Method runs very well on GPUs.
- ▶ Still an active research area.



ECM is very efficient at factoring random numbers, once small factors are removed.

Heuristic running time $L_p(1/2, \sqrt{2}) = O(e^{\sqrt{2}\sqrt{\ln p \ln \ln p}})$.

Quadratic Sieve Intuition: Fermat factorization

Main insight: If we can find two squares a^2 and b^2 such that

$$a^2 \equiv b^2 \pmod{N}$$

Then

$$a^2 - b^2 = (a + b)(a - b) \equiv 0 \pmod{N}$$

and we might hope that one of $a + b$ or $a - b$ shares a nontrivial common factor with N .

Quadratic Sieve Intuition: Fermat factorization

Main insight: If we can find two squares a^2 and b^2 such that

$$a^2 \equiv b^2 \pmod{N}$$

Then

$$a^2 - b^2 = (a + b)(a - b) \equiv 0 \pmod{N}$$

and we might hope that one of $a + b$ or $a - b$ shares a nontrivial common factor with N .

First try:

1. Start at $c = \lceil \sqrt{N} \rceil$
2. Check $c^2 - N, (c + 1)^2 - N, \dots$ until we find a square.

This is Fermat factorization, which could take up to p steps.

Quadratic Sieve

General-purpose factoring

Intuition

We might not find a square outright, but we can construct a square as a product of numbers we look through.

Quadratic Sieve

General-purpose factoring

Intuition

We might not find a square outright, but we can construct a square as a product of numbers we look through.

1. **Sieving** Try to factor each of $c^2 - N, (c + 1)^2 - N, \dots$
2. Only save a $d_i = c_i^2 - N$ if all of its prime factors are less than some bound B . (If it is *B-smooth*.)
3. Store each d_i by its exponent vector $d_i = 2^{e_2} 3^{e_3} \dots B^{e_B}$.
4. If $\prod_i d_i$ is a square, then its exponent vector contains only even entries.

Quadratic Sieve

General-purpose factoring

Intuition

We might not find a square outright, but we can construct a square as a product of numbers we look through.

1. **Sieving** Try to factor each of $c^2 - N, (c + 1)^2 - N, \dots$
2. Only save a $d_i = c_i^2 - N$ if all of its prime factors are less than some bound B . (If it is *B-smooth*.)
3. Store each d_i by its exponent vector $d_i = 2^{e_2} 3^{e_3} \dots B^{e_B}$.
4. If $\prod_i d_i$ is a square, then its exponent vector contains only even entries.
5. **Linear Algebra** Once enough factorizations have been collected, can use linear algebra to find a linear dependence mod 2.

Quadratic Sieve

General-purpose factoring

Intuition

We might not find a square outright, but we can construct a square as a product of numbers we look through.

1. **Sieving** Try to factor each of $c^2 - N, (c + 1)^2 - N, \dots$
2. Only save a $d_i = c_i^2 - N$ if all of its prime factors are less than some bound B . (If it is B -smooth.)
3. Store each d_i by its exponent vector $d_i = 2^{e_2} 3^{e_3} \dots B^{e_B}$.
4. If $\prod_i d_i$ is a square, then its exponent vector contains only even entries.
5. **Linear Algebra** Once enough factorizations have been collected, can use linear algebra to find a linear dependence mod 2.
6. **Square roots** Take square roots and hope for a nontrivial factorization. Math exercise: Square product has 50% chance of factoring pq .

An example of the quadratic sieve

Let's try to factor $N = 2759$.

An example of the quadratic sieve

Let's try to factor $N = 2759$.

Sieving values $(\lceil \sqrt{N} + i \rceil)^2 \bmod N$:

An example of the quadratic sieve

Let's try to factor $N = 2759$.

Sieving values $(\lceil \sqrt{N} + i \rceil)^2 \bmod N$:

$$53^2 - 2759 = 50 = 2 \cdot 5^2.$$

An example of the quadratic sieve

Let's try to factor $N = 2759$.

Sieving values $(\lceil \sqrt{N} + i \rceil)^2 \bmod N$:

$$53^2 - 2759 = 50 = 2 \cdot 5^2.$$

$$54^2 - 2759 = 157.$$

An example of the quadratic sieve

Let's try to factor $N = 2759$.

Sieving values $(\lceil \sqrt{N} + i \rceil)^2 \bmod N$:

$$53^2 - 2759 = 50 = 2 \cdot 5^2.$$

$$54^2 - 2759 = 157.$$

$$55^2 - 2759 = 266.$$

An example of the quadratic sieve

Let's try to factor $N = 2759$.

Sieving values $(\lceil \sqrt{N} + i \rceil)^2 \bmod N$:

$$53^2 - 2759 = 50 = 2 \cdot 5^2.$$

$$54^2 - 2759 = 157.$$

$$55^2 - 2759 = 266.$$

$$56^2 - 2759 = 377.$$

An example of the quadratic sieve

Let's try to factor $N = 2759$.

Sieving values $(\lceil \sqrt{N} + i \rceil)^2 \bmod N$:

$$53^2 - 2759 = 50 = 2 \cdot 5^2.$$

$$54^2 - 2759 = 157.$$

$$55^2 - 2759 = 266.$$

$$56^2 - 2759 = 377.$$

$$57^2 - 2759 = 490 = 2 \cdot 5 \cdot 7^2.$$

An example of the quadratic sieve

Let's try to factor $N = 2759$.

Sieving values $(\lceil \sqrt{N} + i \rceil)^2 \bmod N$:

$$53^2 - 2759 = 50 = 2 \cdot 5^2.$$

$$54^2 - 2759 = 157.$$

$$55^2 - 2759 = 266.$$

$$56^2 - 2759 = 377.$$

$$57^2 - 2759 = 490 = 2 \cdot 5 \cdot 7^2.$$

$$58^2 - 2759 = 605 = 5 \cdot 11^2.$$

An example of the quadratic sieve

Let's try to factor $N = 2759$.

Sieving values $(\lceil \sqrt{N} + i \rceil)^2 \bmod N$:

$$53^2 - 2759 = 50 = 2 \cdot 5^2.$$

$$54^2 - 2759 = 157.$$

$$55^2 - 2759 = 266.$$

$$56^2 - 2759 = 377.$$

$$57^2 - 2759 = 490 = 2 \cdot 5 \cdot 7^2.$$

$$58^2 - 2759 = 605 = 5 \cdot 11^2.$$

Linear Algebra: The *product* $50 \cdot 490 \cdot 605$ is a square:

$$2^2 \cdot 5^4 \cdot 7^2 \cdot 11^2.$$

An example of the quadratic sieve

Let's try to factor $N = 2759$.

Sieving values $(\lceil \sqrt{N} + i \rceil)^2 \bmod N$:

$$53^2 - 2759 = 50 = 2 \cdot 5^2.$$

$$54^2 - 2759 = 157.$$

$$55^2 - 2759 = 266.$$

$$56^2 - 2759 = 377.$$

$$57^2 - 2759 = 490 = 2 \cdot 5 \cdot 7^2.$$

$$58^2 - 2759 = 605 = 5 \cdot 11^2.$$

Linear Algebra: The *product* $50 \cdot 490 \cdot 605$ is a square:

$$2^2 \cdot 5^4 \cdot 7^2 \cdot 11^2.$$

Recall idea: If $a^2 - N$ is a square b^2 then $N = (a - b)(a + b)$.

QS computes $\gcd\{2759, 53 \cdot 57 \cdot 58 - \sqrt{50 \cdot 490 \cdot 605}\} = 31$.

Quadratic Sieve running time

- ▶ How do we choose B ?
- ▶ How many numbers do we have to try to factor?
- ▶ Depends on (heuristic) probability that a randomly chosen number is B -smooth.

Running time: $L_N(1/2, 1) = e^{(1+o(1))\sqrt{\ln N \ln \ln N}}$.

Number field sieve

Best running time for general purpose factoring

Insight

- ▶ Replace relationship $a^2 = b^2 \bmod N$ with a homomorphism between ring of integers \mathcal{O}_K in a specially chosen number field and \mathbb{Z}_N .

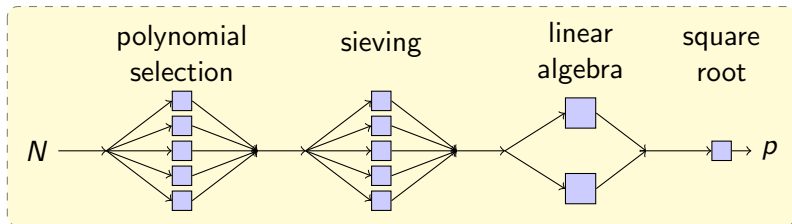
$$\varphi : \mathcal{O}_K \mapsto \mathbb{Z}_N$$

Algorithm

1. **Polynomial selection** Find a good choice of number field K .
2. **Relation finding** Factor elements over \mathcal{O}_K and over \mathbb{Z} .
3. **Linear algebra** Find a square in \mathcal{O}_K and a square in \mathbb{Z}
4. **Square roots** Take square roots, map into \mathbb{Z} , and hope we find a factor.

Running time: $L_N(1/3, \sqrt[3]{64/9}) = e^{(1.923+o(1))(\ln N)^{1/3}(\ln \ln N)^{2/3}}$.

Running the NFS with CADO-NFS



	Sieving			Linear Algebra	
	I	1pb	core-years	rows	core-years
RSA-512	14	29	0.5	4.3M	0.33

Times for 75 c4.8xlarge Amazon ec2 instances:

	polysel		sieving		linalg		sqrt	
	2400 cores		36 cores		36 cores		36 cores	
RSA-512	1.5 hours		2.3 hours		3 hours		5 mins	

Does anyone *use* 512-bit RSA?

International Traffic in Arms Regulations

April 1, 1992 version

Category XIII--Auxiliary Military Equipment ...

(b) Information Security Systems and equipment, cryptographic devices, software, and components specifically designed or modified therefore, including:

(1) Cryptographic (including key management) systems, equipment, assemblies, modules, integrated circuits, components or software with the capability of maintaining secrecy or confidentiality of information or information systems, except cryptographic equipment and software as follows:

(i) Restricted to decryption functions specifically designed to allow the execution of copy protected software, provided the decryption functions are not user-accessible.

(ii) Specially designed, developed or modified for use in machines for banking or money transactions, and restricted to use only in such transactions. Machines for banking or money transactions include automatic teller machines, self-service statement printers, point of sale terminals or equipment for the encryption of interbanking transactions.

...

Bernstein v. US

(1) CJ 191-92

61. On or about June 30, 1992, Plaintiff submitted a CJ Request to Defendant STATE DEPARTMENT to determine whether publication of 1) the paper entitled "The Snuffle Encryption System," 2) source code for the encryption portion of Snuffle, and 3) source code for the decryption portion of Snuffle required a license under the ITAR. Filed under seal herewith as Exhibit "A" is a true and correct copy of the cover letter accompanying CJ 191-92.

62. Plaintiff is informed and believes and based upon such information and belief alleges that his request, labelled CJ 191-92 by the Defendant STATE DEPARTMENT, was referred to, among others, Defendants MARK KORO and GREG STARK acting under color of authority of Defendant NATIONAL SECURITY AGENCY for determination of whether a license was required prior to publication of the Items.

63. On or about August 20, 1992, Defendant WILLIAM G. ROBINSON, acting under color of authority of Defendant STATE DEPARTMENT, informed Plaintiff that he would need a license in order to publish the items included in CJ 191-92. Attached hereto as Exhibit "B" is a true and correct copy of Defendant ROBINSON's letter to Plaintiff.

Commerce Control List: Category 5 - Info. Security

(From 2014)

a.1.a. A symmetric algorithm employing a key length in excess of 56-bits; or

a.1.b. An asymmetric algorithm where the security of the algorithm is based on any of the following:

a.1.b.1. Factorization of integers in excess of 512 bits (e.g., RSA);

a.1.b.2. Computation of discrete logarithms in a multiplicative group of a finite field of size greater than 512 bits (e.g., Diffie-Hellman over $\mathbb{Z}/p\mathbb{Z}$); or

a.1.b.3. Discrete logarithms in a group other than mentioned in 5A002.a.1.b.2 in excess of 112 bits (e.g., Diffie-Hellman over an elliptic curve);

a.2. Designed or modified to perform cryptanalytic functions;

Personal

Small Business

Wealth Management

Businesses & Institutions



Bank of America

Locations : Contact Us : Help : En español

Search Bank of America

Online ID

Sign In



is Online ID

Enroll

ount location

Bank

Borrow

Invest



Protect

Pla

\$100
bonus cash back offer



Offer Details

BankAmericard Cash Rewards™ credit card

1% cash back everywhere, every time

2% cash back on groceries

3% cash back on gas

Grocery/gas bonus rewards on \$1,500 in combined purchases each quarter.

for: Select a state

Go

[Website Ad](#)

Banking

Secure access to your money anytime, anywhere.

Online Bill Pay



The fast convenient way to pay your bills.

Donating homes to veterans



We've committed to donate 1000 properties to veterans and first responders.

Locations

Enter city, state or ZIP code

[More search options](#)

Other services

of America 

Online ID

Sign In

is Online ID

ount location




\$10

bonus cash

for:

Banking

Secure access to
your money anytime,
anywhere.

 VeriSign Class 3 Public Primary Certification Authority - G5 VeriSign Class 3 Extended Validation SSL CA www.bankofamerica.comCommon Name www.bankofamerica.com

Issuer Name

Country US

Organization VeriSign, Inc.

Organizational Unit VeriSign Trust Network

Organizational Unit Terms of use at <https://www.verisign.com/rpa> (c)06

Common Name VeriSign Class 3 Extended Validation SSL CA

Serial Number 77 24 50 6D 4F 9A 87 9D 4B C6 6E 67 88 F2 60 C9

Version 3

Signature Algorithm SHA-1 with RSA Encryption (1.2.840.113549.1.1.5)

Parameters none

Not Valid Before Tuesday, February 28, 2012 7:00:00 PM Eastern Standard Time

Not Valid After Thursday, February 28, 2013 6:59:59 PM Eastern Standard Time

Public Key Info

Algorithm RSA Encryption (1.2.840.113549.1.1.1)

Parameters none

Public Key 256 bytes : BD E6 52 EB 6A 9D C5 B3 ...

Exponent 65537

Key Size 2048 bits

Key Usage Encrypt, Verify, Wrap, Derive

Signature 256 bytes : 77 D6 C8 64 DC 24 3F 8C ...

Businesses & Institutions

añol

Protect

Pla

Americard Cash

ds™ credit card

ck everywhere, every tim

ck on groceries

ck on gas

s rewards on \$1,500 in combined
quarter.[Website Ad](#)

Locations

[More search options](#)

Other services

Export cipher suites in TLS

TLS_RSA_EXPORT_WITH_RC4_40_MD5
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_Annon_EXPORT_WITH_RC4_40_MD5
TLS_DH_Annon_EXPORT_WITH_DES40_CBC_SHA

In March 2015, export cipher suites supported by 36.7% of the 14 million sites serving browser-trusted certificates!

FREAK attack [BDFKPSZZ 2015]: Use fast 512-bit factorization to downgrade modern browsers to broken export-grade RSA.



New Directions in Cryptography

Invited Paper

WHITFIELD DIFFIE AND MARTIN E. HELLMAN, MEMBER, IEEE

Textbook Diffie-Hellman

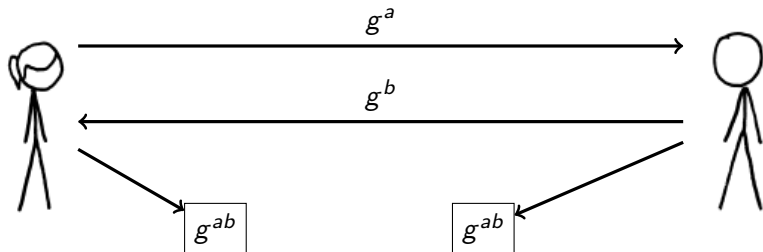
[Diffie Hellman 1976]

Public Parameters

G a cyclic group (e.g. \mathbb{F}_p^* , or an elliptic curve)

g group generator

Key Exchange



Computational problems

Discrete Log

Problem: Given g^a , compute a .

- ▶ Solving this problem permits attacker to compute shared key by computing a and raising $(g^b)^a$.
- ▶ Discrete log is in NP and coNP \rightarrow not NP-complete (unless $P=NP$ or similar).

Computational problems

Diffie-Hellman problem

Problem: Given g^a , g^b , compute g^{ab} .

- ▶ Exactly problem of computing shared key from public information.
- ▶ Reduces to discrete log in some cases:
 - ▶ “Diffie-Hellman is as strong as discrete log for certain primes” [den Boer 1988] “both problems are (probabilistically) polynomial-time equivalent if the totient of $p - 1$ has only small prime factors”
 - ▶ “Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms” [Maurer 1994] “if ... an elliptic curve with smooth order can be constructed efficiently, then ... [the discrete log] can be reduced efficiently to breaking the Diffie-Hellman protocol”
- ▶ (Computational) Diffie-Hellman assumption: This problem is hard in general.

FIPS PUB 186-3

**FEDERAL INFORMATION PROCESSING STANDARDS
PUBLICATION**

Digital Signature Standard (DSS)

CATEGORY: COMPUTER SECURITY

SUBCATEGORY: CRYPTOGRAPHY

The DSA Algorithm

DSA Public Key

p prime

q prime, divides $(p - 1)$

g generator of subgroup of
order $q \bmod p$

$$y = g^x \bmod p$$

Private Key

x private key

Verify

$$u_1 = H(m)s^{-1} \bmod q$$

$$u_2 = rs^{-1} \bmod q$$

$$r \stackrel{?}{=} g^{u_1} y^{u_2} \bmod p \bmod q$$

Sign

Generate random k .

$$r = g^k \bmod p \bmod q$$

$$s = k^{-1}(H(m) + xr) \bmod q$$

Computational problems

Discrete Log

- ▶ Breaking DSA is equivalent to computing discrete logs in the random oracle model. [Pointcheval, Vaudenay 96]

Discrete log algorithms

Three families of discrete log algorithms:

1. Algorithms whose running time depends on the size of the order of the subgroup.
 - ▶ Good for computing discrete logs in subgroups of small or smooth order.
2. Algorithms whose running time depends on the size of the log.
 - ▶ Good for computing discrete logs in a known small interval.
3. Algorithms whose running time depends on the size of the modulus.
 - ▶ Good for computing discrete logs in subgroups of large order.

Computing Discrete Logs in $O(\sqrt{q})$ time

Goal: Solve $g^\ell \equiv t \pmod{p}$. g has order q .

Baby-Step Giant-Step Algorithm

1. Compute $g^0, g^{\lfloor \sqrt{q} \rfloor}, g^{2\lfloor \sqrt{q} \rfloor}, \dots$. “Giant steps”

$$g^{0\lfloor \sqrt{q} \rfloor} \quad g^{1\lfloor \sqrt{q} \rfloor} \quad g^{2\lfloor \sqrt{q} \rfloor} \quad \overset{t}{\circ} \quad g^{3\lfloor \sqrt{q} \rfloor} \quad g^{4\lfloor \sqrt{q} \rfloor}$$

Computing Discrete Logs in $O(\sqrt{q})$ time

Goal: Solve $g^\ell \equiv t \pmod{p}$. g has order q .

Baby-Step Giant-Step Algorithm

1. Compute $g^0, g^{\lfloor \sqrt{q} \rfloor}, g^{2\lfloor \sqrt{q} \rfloor}, \dots$. “Giant steps”

A number line representing the discrete logarithm space from 0 to q . Vertical tick marks are placed at intervals of $\lfloor \sqrt{q} \rfloor$. Below the line, the labels $g^0, g^{\lfloor \sqrt{q} \rfloor}, g^{2\lfloor \sqrt{q} \rfloor}, g^{3\lfloor \sqrt{q} \rfloor}, g^{4\lfloor \sqrt{q} \rfloor}$ are aligned with their respective tick marks. Above the line, a red circle labeled t is positioned between the tick marks for $g^{2\lfloor \sqrt{q} \rfloor}$ and $g^{3\lfloor \sqrt{q} \rfloor}$.

2. Compute tg^1, tg^2, \dots until hit “giant step”. “Baby steps”

A number line representing the discrete logarithm space from 0 to q . Vertical tick marks are placed at intervals of $\lfloor \sqrt{q} \rfloor$. Below the line, the labels $g^0, g^{\lfloor \sqrt{q} \rfloor}, g^{2\lfloor \sqrt{q} \rfloor}, g^{3\lfloor \sqrt{q} \rfloor}, g^{4\lfloor \sqrt{q} \rfloor}$ are aligned with their respective tick marks. Above the line, a blue circle labeled tg^5 is positioned between the tick marks for $g^{2\lfloor \sqrt{q} \rfloor}$ and $g^{3\lfloor \sqrt{q} \rfloor}$. A red circle labeled t is positioned above the tick mark for $g^{2\lfloor \sqrt{q} \rfloor}$.

Computing Discrete Logs in $O(\sqrt{q})$ time

Goal: Solve $g^\ell \equiv t \pmod{p}$. g has order q .

Baby-Step Giant-Step Algorithm

1. Compute $g^0, g^{\lfloor \sqrt{q} \rfloor}, g^{2\lfloor \sqrt{q} \rfloor}, \dots$. “Giant steps”

A number line representing the group elements $g^0, g^{\lfloor \sqrt{q} \rfloor}, g^{2\lfloor \sqrt{q} \rfloor}, g^{3\lfloor \sqrt{q} \rfloor}, g^{4\lfloor \sqrt{q} \rfloor}, \dots$. The elements are marked with vertical tick marks and labeled below. Above the line, a red circle labeled t is positioned between $g^{2\lfloor \sqrt{q} \rfloor}$ and $g^{3\lfloor \sqrt{q} \rfloor}$, indicating it is not a giant step.

2. Compute tg^1, tg^2, \dots until hit “giant step”. “Baby steps”

A number line representing the group elements $g^0, g^{\lfloor \sqrt{q} \rfloor}, g^{2\lfloor \sqrt{q} \rfloor}, g^{3\lfloor \sqrt{q} \rfloor}, g^{4\lfloor \sqrt{q} \rfloor}, \dots$. The elements are marked with vertical tick marks and labeled below. Above the line, a blue circle labeled tg^5 is positioned directly above the tick mark for $g^{3\lfloor \sqrt{q} \rfloor}$, indicating a collision. A red circle labeled t is positioned above the line between $g^{2\lfloor \sqrt{q} \rfloor}$ and $g^{3\lfloor \sqrt{q} \rfloor}$.

3. Solve for t from collision: $tg^5 = g^{3\lfloor \sqrt{q} \rfloor}$

► Also works for finding ℓ in known interval.

Pollard rho for discrete log

- ▶ Can also use Pollard rho idea to compute discrete logs.
- ▶ Take random walk; terminates in $O(\sqrt{q})$ time. Reduces storage requirement.
- ▶ Need to use a different random function. Pollard suggested

$$f(x) = \begin{cases} gx & 1 \leq x < p/3 \\ x^2 & p/3 \leq x < 2p/3 \\ yx & 2p/3 \leq x < p \end{cases}$$

Using more intervals works better.

Taking advantage of subgroups: Pohlig-Hellman

- ▶ If $q = \prod_i q_i^{e_i}$:
 1. Can solve discrete log in subgroup of order q_i in time $\sqrt{q_i}$.
 2. Can solve discrete log in each subgroup of order $q_i^{e_i}$ in time $e_i \sqrt{q_i}$.
 3. Can use Chinese remainder theorem to reconstruct log mod q .

Best practice: To avoid attacks, choose group so that g generates subgroup of prime order $q > 2^{160}$.

Short exponents with smooth-order primes: A sad tale

[van Oorschot, Wiener]

In recent Logjam work [ABDGGHHSTVVWZZ 2015], we scanned entire internet to study Diffie-Hellman usage in HTTPS.

- ▶ Found 4800 groups (p, g) where $(p - 1)/2$ was not prime.
- ▶ Applied ECM to opportunistically factor $(p - 1)/2$.
- ▶ Learned prime factors of order of g for 750 groups, used in 40,000 connections across our Internet scans.

Short exponents with smooth-order primes: A sad tale

[van Oorschot, Wiener]

In recent Logjam work [ABDGGHHSTVVWZZ 2015], we scanned entire internet to study Diffie-Hellman usage in HTTPS.

- ▶ Found 4800 groups (p, g) where $(p - 1)/2$ was not prime.
- ▶ Applied ECM to opportunistically factor $(p - 1)/2$.
- ▶ Learned prime factors of order of g for 750 groups, used in 40,000 connections across our Internet scans.
- ▶ Some implementations use short exponents x in g^x of length 128 or 160 bits.
- ▶ If $x < \prod_i q_i^{e_i}$ for $q_i^{e_i}$ dividing order of g , can use Pollard rho+CRT to recover x in $\max_i \sqrt{q_i}$ time.

Short exponents with smooth-order primes: A sad tale

[van Oorschot, Wiener]

In recent Logjam work [ABDGGHHSTVVWZZ 2015], we scanned entire internet to study Diffie-Hellman usage in HTTPS.

- ▶ Found 4800 groups (p, g) where $(p - 1)/2$ was not prime.
- ▶ Applied ECM to opportunistically factor $(p - 1)/2$.
- ▶ Learned prime factors of order of g for 750 groups, used in 40,000 connections across our Internet scans.
- ▶ Some implementations use short exponents x in g^x of length 128 or 160 bits.
- ▶ If $x < \prod_i q_i^{e_i}$ for $q_i^{e_i}$ dividing order of g , can use Pollard rho+CRT to recover x in $\max_i \sqrt{q_i}$ time.
- ▶ Implemented Pohlig-Hellman algorithm to test if servers used short exponents; computed information about secret exponent in 460 exchanges, and whole exponent for 159 hosts.

Subexponential-time algorithms: Index calculus

Goal: Solve $g^\ell \equiv t \pmod{p}$.

Fix some a priori smoothness bound B .

Subexponential-time algorithms: Index calculus

Goal: Solve $g^\ell \equiv t \pmod{p}$.

Fix some a priori smoothness bound B .

1. **Relation finding:** Enumerate pairs of B -smooth integers equivalent mod p .

$$\begin{aligned} p_1^{a_{11}} \dots B^{a_{1k}} &= 1 \equiv p + 1 = p_1^{r_{11}} p_2^{r_{12}} \dots B^{r_{1k}} \\ p_1^{a_{21}} \dots B^{a_{2k}} &= 2 \equiv p + 2 = p_1^{r_{21}} p_2^{r_{22}} \dots B^{r_{2k}} \\ &\vdots \\ p_1^{a_{k1}} \dots B^{a_{kk}} &= z \equiv p + z = p_1^{r_{k1}} p_2^{r_{k2}} \dots B^{r_{kk}} \end{aligned}$$

Index calculus: Linear algebra

Take log of both sides. Assume subgroup of order q . Then

$$a_{11} \log p_1 + \cdots + a_{1k} \log p_k \equiv r_{11} \log p_1 + \cdots + r_{1k} \log B \pmod{q}$$

$$a_{21} \log p_1 + \cdots + a_{2k} \log p_k \equiv r_{21} \log p_1 + \cdots + r_{2k} \log B \pmod{q}$$

$$\vdots$$

$$a_{k1} \log p_1 + \cdots + a_{kk} \log p_k \equiv r_{k1} \log p_1 + \cdots + r_{kk} \log B \pmod{q}$$

Also get some relations for free: $\log -1 = (p-1)/2 \dots$

2. Linear Algebra: Solve system of equations for $\log p_i$:

$$\log p_1 \equiv s_1$$

$$\vdots$$

$$\log p_k \equiv s_k$$

Actually computing individual logs

Input target t .

3. Try to find some B -smooth value

$$g^R t = p_1^{e_1} \dots B^{e_B}$$

Then using known values of $\log p_i$ write

$$\log t = -R + e_1 \log p_1 + \dots + e_B \log B \bmod q$$

Index calculus running time

1. **Relation collection** Runtime depends on (1) work to test if integer is B -smooth, (2) probability integer is B -smooth, (3) B .
2. **Linear algebra** Runtime depends on cost of sparse linear algebra for B -dimensional matrix mod q .
3. **Individual log** Runtime depends on probability that $g^R t$ is B -smooth.

Optimizing for B gives runtime of

$$\exp((\sqrt{2} + o(1))\sqrt{\log p \log \log p}) = L_p(1/2, \sqrt{2})$$

Number field sieve

[Gordon], [Joux, Lercier], [Semaev]

1. **Polynomial selection:** Find a polynomial f and an integer m such that $f(m) \equiv 0 \pmod{p}$, $\deg f = 5$ or 6 , coeffs of f relatively small. Defines a number field $\mathbb{Q}(x)/f(x)$.

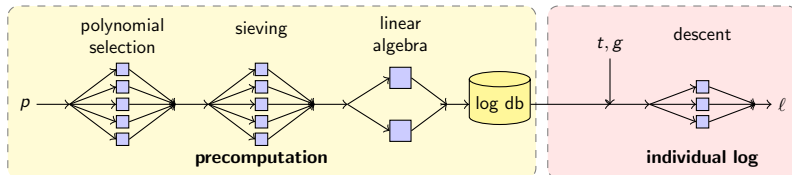
For $\gamma = \sum_i a_i \alpha^i$ in ring of integers, define homomorphism $\varphi(\gamma) = \sum_i a_i m^i$ to $\mathbb{Z}/p\mathbb{Z}$.

2. **Relation collection** Collect relations of form

$$p_1^{a_{11}} \dots p_k^{a_{1k}} = a + b\alpha \equiv a + bm = p_1^{r_{11}} \dots p_k^{r_{1k}}$$

3. **Linear algebra** Once there are enough relations, solve for $\log p_i$.
4. **Individual log** “Descent” Try to write target t as sum of logs in known database.

Implementing the NFS with CADO-NFS



$L(1/3, 1.923)$

$L(1/3, 1.232)$

	Sieving			Linear Algebra		Descent
	I	1pb	core-years	rows	core-years	core-time
RSA-512	14	29	0.5	4.3M	0.33	
DH-512	15	27	2.5	2.1M	7.7	10 mins

Times for cluster computation:

	polysel	sieving	linalg	descent
	2000-3000 cores		288 cores	24 cores
DH-512	3 hours	15 hours	120 hours	90 seconds

www.tue.nl

Your connection to this site is private.

Permissions

Connection



The identity of this website has been verified by TERENA SSL CA 2 but does not have public audit records.

[Certificate Information](#)



Your connection to www.tue.nl is encrypted with modern cryptography.

The connection uses TLS 1.2.

The connection is encrypted and authenticated using AES_128_GCM and uses DHE_RSA as the key exchange mechanism.

**Site information**

You first visited this site on May 22, 2015.

[What do these mean?](#)



Ontmoet een robot tijdens technologisch festival

27 mei 2015

Technische Universiteit Eindhoven organiseert TU/eXperience Day op zondag 31 mei.



Studiegids

Intranet

A A

Contact



Zoek

TU/eXperien

jouw droom **X**onze



Agenda



Afscheidscollege prof.ir. Ger Maas

29 mei 2015

TU/eXperience Day 2015

31 mei 2015

Hajraa Buitentoernooi 2015

12 juni 2015 t/m 14 juni 2015

Afscheidscollege prof.dr.ir. Pieter Wijn

12 juni 2015

3TU.SAI diploma-uitreiking

16 juni 2015

[Meer activiteiten](#)



Studenten tevreden over opleidingen TU/e

22 mei 2015

Studenten beoordelen hun TU/e-opleiding gemiddeld met een 3.8, op een schaal van 5. Dat blijkt uit de Nationale Studenten Enquête 2015.

Direct naar

- ▶ [Route en plattegrond](#)
- ▶ [Video: dit is TU/e](#)
- ▶ [Vacatures](#)
- ▶ [Bibliotheek](#)
- ▶ [Promotie-agenda](#)
- ▶ [Feiten en cijfers](#)
- ▶ [Nieuwssite Cursor](#)



Does anyone *use* 512-bit Diffie-Hellman?

Export cipher suites in TLS

```
TLS_RSA_EXPORT_WITH_RC4_40_MD5  
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5  
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA  
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA  
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA  
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA  
TLS_DH_Annon_EXPORT_WITH_RC4_40_MD5  
TLS_DH_Annon_EXPORT_WITH_DES40_CBC_SHA
```

In April 2015, DHE_EXPORT cipher suites supported by 8.4% of the Alexa top 1 Million!

Logjam attack [ABDGGHHSTVWZZ 2015]: Use fast 512-bit discrete log in fixed groups to downgrade HTTPS connections to insecure DHE_EXPORT cipher suites.

Scaling discrete log computations to larger key sizes

	<i>Vulnerable servers, if the attacker can precompute for . . .</i>			
	all 512-bit p	all 768-bit p	one 1024-bit p	ten 1024-bit p
HTTPS Top 1M MITM	45K (8.4%)	45K (8.4%)	205K (37.1%)	309K (56.1%)
HTTPS Top 1M	118 (0.0%)	407 (0.1%)	98.5K (17.9%)	132K (24.0%)
HTTPS Trusted MITM	489K (3.4%)	556K (3.9%)	1.84M (12.8%)	3.41M (23.8%)
HTTPS Trusted	1K (0.0%)	46.7K (0.3%)	939K (6.56%)	1.43M (10.0%)
IKEv1 IPv4	–	64K (2.6%)	1.69M (66.1%)	1.69M (66.1%)
IKEv2 IPv4	–	66K (5.8%)	726K (63.9%)	726K (63.9%)
SSH IPv4	–	–	3.6M (25.7%)	3.6M (25.7%)

Summary

- ▶ RSA and “mod- p ” Diffie-Hellman are old and busted.
- ▶ (At least for ≤ 1024 -bit keys.)
- ▶ Elliptic curves are the new hotness. (See Tanja’s talk!)