Imperfect Forward Secrecy
How Diffie-Hellman Fails in Practice

David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, 
weakdh.org

July 7, 2015
New Directions in Cryptography

Invited Paper

WHITFIELD DIFFIE AND MARTIN E. HELLMAN, MEMBER, IEEE
Textbook Diffie-Hellman
[Diffie Hellman 1976]

Public Parameters

$G$ a group (e.g. $\mathbb{F}_p^*$)

$g$ group generator

Key Exchange
Reminders: Orders and groups

For our purposes, $g$ generates multiplicative group mod $p$.

**Theorem (Fermat’s Little Theorem)**

$g^{p-1} \equiv 1 \mod p$ for any $0 < g < p$.

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

**Theorem (Lagrange)**

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

**Discrete Log**

**Problem:** Given $t = g^\ell$, compute $\ell$. 
Warm-up: Elementary discrete log algorithms

Pollard rho, baby step giant step

- $O(\sqrt{q})$ for (sub)group of order $q$. 

Pohlig-Hellman

1. Factor group order $q = \prod q^e_i$.
2. Solve discrete log in each subgroup in time $e_i \sqrt{q_i}$.
3. Use Chinese remainder theorem to reconstruct log mod $q$.

Best practice: $g$ should generate large prime-order subgroup mod $p$. Common choices include "safe" primes $p = 2q + 1$ or DSA groups.
Warm-up: Elementary discrete log algorithms

Pollard rho, baby step giant step

- \(O(\sqrt{q})\) for (sub)group of order \(q\).

Pollard lambda

- \(O(\sqrt{r})\) for exponent in size range \(r\).

Pohlig-Hellman

1. Factor group order \(q = \prod q_i^{e_i}\).
2. Solve discrete log in each subgroup in time \(e_i \sqrt{q_i}\).
3. Use Chinese remainder theorem to reconstruct log mod \(q\).

Best practice: \(g\) should generate large prime-order subgroup mod \(p\). Common choices include "safe" primes \(p = 2q + 1\) or DSA groups.
Warm-up: Elementary discrete log algorithms

Pollard rho, baby step giant step

- $O(\sqrt{q})$ for (sub)group of order $q$.

Pollard lambda

- $O(\sqrt{r})$ for exponent in size range $r$.

Pohlig-Hellman

1. Factor group order $q = \prod_i q_i^{e_i}$.
2. Solve discrete log in each subgroup in time $e_i \sqrt{q_i}$.
3. Use Chinese remainder theorem to reconstruct log mod $q$. 

Best practice: $g$ should generate large prime-order subgroup mod $p$. Common choices include “safe” primes $p = 2q + 1$ or DSA groups.
Warm-up: Elementary discrete log algorithms

Pollard rho, baby step giant step

- $O(\sqrt{q})$ for (sub)group of order $q$.

Pollard lambda

- $O(\sqrt{r})$ for exponent in size range $r$.

Pohlig-Hellman

1. Factor group order $q = \prod_i q_i^{e_i}$.
2. Solve discrete log in each subgroup in time $e_i \sqrt{q_i}$.
3. Use Chinese remainder theorem to reconstruct log mod $q$.

Best practice: $g$ should generate large prime-order subgroup mod $p$. Common choices include “safe” primes $p = 2q + 1$ or DSA groups.
Pop quiz

Do people deploying cryptography follow best practice?
Diffie-Hellman deployment in practice

- 23.9% of 14.3M HTTPS servers support Diffie-Hellman
Diffie-Hellman deployment in practice

- 23.9% of 14.3M HTTPS servers support Diffie-Hellman
- Observed 70,000 distinct primes $p$

...
Diffie-Hellman deployment in practice

- 23.9% of 14.3M HTTPS servers support Diffie-Hellman
- Observed 70,000 distinct primes \( p \)
- Found 4800 groups \((p, g)\) where \((p - 1)/2\) was not prime.
Diffie-Hellman deployment in practice

- 23.9% of 14.3M HTTPS servers support Diffie-Hellman
- Observed 70,000 distinct primes $p$
- Found 4800 groups $(p, g)$ where $(p - 1)/2$ was not prime.
- Applied ECM to opportunistically factor $(p - 1)/2$. 
Diffie-Hellman deployment in practice

- 23.9% of 14.3M HTTPS servers support Diffie-Hellman
- Observed 70,000 distinct primes $p$
- 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.
Diffie-Hellman deployment in practice

- 23.9% of 14.3M HTTPS servers support Diffie-Hellman
- Observed 70,000 distinct primes $p$
- 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.

For random $p$, $p - 1$ likely to have large factors, so might not recover full discrete log.
Some implementations use short exponents $\ell$: 128, 160 bits.
Some implementations use short exponents $\ell$: 128, 160 bits.

If $\ell < \prod_i q_i^{e_i}$ for $q_i^{e_i} \mid \operatorname{ord}_p(g)$, Pohlig-Hellman over only these subgroups reconstructs $\ell$ by CRT.
Some implementations use short exponents $\ell$: 128, 160 bits.

If $\ell < \prod_i q_i^{e_i}$ for $q_i^{e_i} \mid \text{ord}_{p}(g)$, Pohlig-Hellman over only these subgroups reconstructs $\ell$ by CRT.

Using this attack we computed secret exponent for 159 exchanges and partial information in 460 exchanges.
Goal: Solve \( g^\ell \equiv t \mod p \).

Definition: An integer is \( B \)-smooth if all its prime factors are \( \leq B \). Fix some a priori bound \( B \).
Subexponential algorithms for prime fields: Index calculus

Goal: Solve $g^\ell \equiv t \mod p$.

Definition: An integer is $B$-smooth if all its prime factors are $\leq B$. Fix some a priori bound $B$.

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

\[
\begin{align*}
p_1^{a_{11}} \ldots B^{a_{1k}} &= 1 \equiv p + 1 = p_1^{r_{11}} p_2^{r_{12}} \ldots B^{r_{1k}} \\
p_1^{a_{21}} \ldots B^{a_{2k}} &= 2 \equiv p + 2 = p_1^{r_{21}} p_2^{r_{22}} \ldots B^{r_{2k}} \\
&\vdots \\
p_1^{a_{kk}} \ldots B^{a_{kk}} &= z \equiv p + z = p_1^{r_{k1}} p_2^{r_{k2}} \ldots B^{r_{kk}}
\end{align*}
\]
Index calculus: Linear algebra

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

$$a_{11} \log p_1 + \cdots + a_{1k} \log B \equiv r_{11} \log p_1 + \cdots + r_{1k} \log B \mod q$$
$$a_{21} \log p_1 + \cdots + a_{2k} \log B \equiv r_{21} \log p_1 + \cdots + r_{2k} \log B \mod q$$
$$\vdots$$
$$a_{k1} \log p_1 + \cdots + a_{kk} \log B \equiv r_{k1} \log p_1 + \cdots + r_{kk} \log B \mod q$$

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

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

\[ g^R t = p_1^{e_1} \cdots B^{e_B} \]

Then using known values of \( \log p_i \) write

\[ \log t = -R + e_1 \log p_1 + \cdots + e_B \log B \mod q \]
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

$$\exp((\sqrt{2} + o(1)) \sqrt{\log p \log \log p}) = L_p(1/2, \sqrt{2})$$
1. **Polynomial selection**: Find a polynomial $f$ and an integer $m$ such that $f(m) \equiv 0 \mod p$, 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

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

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

\[ L(1/3, 1.923) \quad \text{and} \quad L(1/3, 1.232) \]
Implementing the NFS with CADO-NFS

\[ L\left(\frac{1}{3}, 1.923\right) \quad \text{and} \quad L\left(\frac{1}{3}, 1.232\right) \]

<table>
<thead>
<tr>
<th></th>
<th>Sieving</th>
<th>Linear Algebra</th>
<th>Descent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I</td>
<td>lpb</td>
<td>core-years</td>
</tr>
<tr>
<td>RSA-512</td>
<td>14</td>
<td>29</td>
<td>0.5</td>
</tr>
<tr>
<td>DH-512</td>
<td>15</td>
<td>27</td>
<td>2.5</td>
</tr>
</tbody>
</table>
Implementing the NFS with CADO-NFS

\[ L(1/3, 1.923) \quad \text{and} \quad L(1/3, 1.232) \]

<table>
<thead>
<tr>
<th></th>
<th>Sieving</th>
<th>Linear Algebra</th>
<th>Descent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( I )</td>
<td>( 1 \text{pb} )</td>
<td>core-years</td>
</tr>
<tr>
<td>RSA-512</td>
<td>14</td>
<td>29</td>
<td>0.5</td>
</tr>
<tr>
<td>DH-512</td>
<td>15</td>
<td>27</td>
<td>2.5</td>
</tr>
</tbody>
</table>

Times for cluster computation:

<table>
<thead>
<tr>
<th></th>
<th>polysel</th>
<th>sieving</th>
<th>linalg</th>
<th>descent</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000-3000 cores</td>
<td>3 hours</td>
<td>15 hours</td>
<td>120 hours</td>
<td>90 seconds</td>
</tr>
<tr>
<td>DH-512</td>
<td>3 hours</td>
<td>15 hours</td>
<td>120 hours</td>
<td>90 seconds</td>
</tr>
</tbody>
</table>
Eli Yablonovitch has been awarded the Isaac Newton Medal by the Institute of Physics in London. This award is the institute’s highest honor and is given to any physicist, regardless of subject area, background or nationality for outstanding contributions to physics.

Monday, July 6
Talk: Challenges in P2P Botnet Monitoring
Shankar Karuppayah, TU Darmstadt / CASED
1-2 p.m., 380 Soda Hall

Friday, July 24
An Empirical Understanding of Conflict-Driven Clause-Learning SAT Solvers
Vijay Ganesh, University of Waterloo
12-1 p.m., 373 Soda Hall

Thursday, August 27
Airbnb Info Session/Tech Talk
5-8 p.m., Wozniak Lounge/430/438 Soda Hall

Friday, August 28
Palantir Technologies Info Session/Tech Talk
5-8 p.m., Wozniak Lounge/430/438 Soda Hall
TLS Diffie-Hellman Key Exchange

**TLS = Transport Layer Security**

- **Hello, 28 byte client random, 4 byte time**
- **List of cipher suites**
TLS Diffie-Hellman Key Exchange

TLS = Transport Layer Security

- hello, 28 byte client random, 4 byte time
- list of cipher suites
- hello, 28 byte server random, 4 byte time
- certificate = public RSA key + CA signatures
- chosen cipher suite, $g^a$, $\text{Sign}_{\text{RSA key}}(g^a)$
TLS Diffie-Hellman Key Exchange

TLS = Transport Layer Security

----

hello, 28 byte client random, 4 byte time

list of cipher suites

hello, 28 byte server random, 4 byte time

certificate = public RSA key + CA signatures

chosen cipher suite, \(g^a\), \(\text{Sign}_{\text{RSAkey}}(g^a)\)

\(g^b\)

\(F(g^{ab}, \text{randoms}) \rightarrow k_m, k_e\)
TLS Diffie-Hellman Key Exchange

TLS = Transport Layer Security

hello, 28 byte client random, 4 byte time

list of cipher suites

hello, 28 byte server random, 4 byte time

certificate = public RSA key + CA signatures

chosen cipher suite, $g^a$, $\text{Sign}_{\text{RSAkey}}(g^a)$

$g^b$

$\text{MAC}_{k_m}(\text{dialog})$

$\text{MAC}_{k_m}(\text{dialog})$

$\text{Enc}_{k_e}(\text{website contents})$

$\text{Enc}_{k_e}(\text{website contents})$
Authentication in TLS Diffie-Hellman Key Exchange

hello, 28 byte client random, 4 byte time

list of cipher suites

hello, 28 byte server random, 4 byte time

certificate = public RSA key + CA signatures

chosen cipher suite, $g^a$, $\text{Sign}_{\text{RSAkey}}(g^a)$

$g^b$

$F(g^{ab}, \text{randoms}) \rightarrow k_m, k_e$

$F(g^{ab}, \text{randoms}) \rightarrow k_m, k_e$

$\text{MAC}_{k_m}(\text{dialog})$

$\text{MAC}_{k_m}(\text{dialog})$

$\text{Enc}_{k_e}(\text{website contents})$
Encryption in TLS Diffie-Hellman Key Exchange

hello, 28 byte client random, 4 byte time

list of cipher suites

hello, 28 byte server random, 4 byte time

certificate = public RSA key + CA signatures

chosen cipher suite, $g^a$, $\text{Sign}_{\text{RSAkey}}(g^a)$

$g^b$

$\text{MAC}_{k_m}(\text{dialog})$

$\text{MAC}_{k_m}(\text{dialog})$

$\text{Enc}_{k_e}(\text{website contents})$
Some differences between practice and theory

- Cipher-suite and parameter negotiation.
- Politics.
- ElGamal is uncommon.
- Group negotiation.
- Short exponents.
- Long-tailed distribution of group parameters.
- Primes with many bits in common.
Does anyone use 512-bit Diffie-Hellman?
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.

...
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;
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_Anon_EXPORT_WITH_RC4_40_MD5`
- `TLS_DH_Anon_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.
Man-in-the-middle downgrade attack to export DH

Protocol flaw: Server does not sign chosen cipher suite.
Carrying out the downgrade attack

- In April 2015, 8.4% of Alexa Top 1M HTTPS domains supported DHE_EXPORT.
- 82% use most common prime. 10% use 2nd most common prime.
- We carried out precomputation for these primes. ≈ 1 week each on 2000-3000 cores.
- Individual descent times on 24-core machine:
Many hosts use the same group parameters.

What about passive attacks for 768 or 1024-bit keys?
## Cost estimates for discrete log

<table>
<thead>
<tr>
<th></th>
<th>Sieving</th>
<th>Linear Algebra</th>
<th>Descent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I</td>
<td>1pb</td>
<td>core-years</td>
</tr>
<tr>
<td>RSA-512</td>
<td>14</td>
<td>29</td>
<td>0.5</td>
</tr>
<tr>
<td>DH-512</td>
<td>15</td>
<td>27</td>
<td>2.5</td>
</tr>
<tr>
<td>RSA-768</td>
<td>16</td>
<td>37</td>
<td>800</td>
</tr>
<tr>
<td>DH-768</td>
<td>17</td>
<td>35</td>
<td>8,000</td>
</tr>
<tr>
<td>RSA-1024</td>
<td>18</td>
<td>42</td>
<td>1,000,000</td>
</tr>
<tr>
<td>DH-1024</td>
<td>19</td>
<td>40</td>
<td>10,000,000</td>
</tr>
</tbody>
</table>

- Use RSA-768 factoring record, adjust to sieve less and modern processors.
- For discrete log, oversieve to decrease linear algebra cost.
- For descent, used early-abort ECM implementation to get experimental timings.
## Cost estimates for discrete log

<table>
<thead>
<tr>
<th></th>
<th>Sieving</th>
<th>Linear Algebra</th>
<th>Descent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I</td>
<td>1pb</td>
<td>core-years</td>
</tr>
<tr>
<td>RSA-512</td>
<td>14</td>
<td>29</td>
<td>0.5</td>
</tr>
<tr>
<td>DH-512</td>
<td>15</td>
<td>27</td>
<td>2.5</td>
</tr>
<tr>
<td>RSA-768</td>
<td>16</td>
<td>37</td>
<td>800</td>
</tr>
<tr>
<td>DH-768</td>
<td>17</td>
<td>35</td>
<td>8,000</td>
</tr>
<tr>
<td>RSA-1024</td>
<td>18</td>
<td>42</td>
<td>1,000,000</td>
</tr>
<tr>
<td>DH-1024</td>
<td>19</td>
<td>40</td>
<td>10,000,000</td>
</tr>
</tbody>
</table>

- [Geiselmann Steinwandt] give ASIC sieving design. 10x their estimates at modern sizes = $8M to sieve 1024-bit DL in one year.
- Titan supercomputer = 300,000 cores → 117 years for linear algebra.
- Assuming 80x speedup for ASIC linear algebra = hundreds of millions of $.
According to another top official also involved with the program, the NSA made an enormous breakthrough several years ago in its ability to cryptanalyze, or break, unfathomably complex encryption systems employed by not only governments around the world but also many average computer users in the US. The upshot, according to this official: “Everybody’s a target; everybody with communication is a target.”

[...]
The breakthrough was enormous, says the former official, and soon afterward the agency pulled the shade down tight on the project, even within the intelligence community and Congress. “Only the chairman and vice chairman and the two staff directors of each intelligence committee were told about it,” he says. The reason? “They were thinking that this computing breakthrough was going to give them the ability to crack current public encryption.”
“Also, we are investing in groundbreaking cryptanalytic capabilities to defeat adversarial cryptography and exploit internet traffic.”
4. Communicate Results

Can we decrypt the VPN traffic?

- If the answer is “No” then explain how to turn it into a “YES!”

- If the answer is “YES!” then…
Happy Dance!!
Turn that Frown Upside Down! From “No” to “YES!”

- Depends on why we couldn’t decrypt it
- Find Pre-Shared Key
- Locate complete paired collect
- Locate both IKE and ESP traffic
- Have collection sites do surveys for the IP’s
- Find better quality collect with rich metadata
IKE Key Exchange for VPNs/IPsec

list of pre-chosen groups

chooses group, $g^a$

$g^b$

$F(g^{ab}, \text{PSK})$

$F(g^{ab}, \text{PSK})$
NSA VPN Attack Orchestration
<table>
<thead>
<tr>
<th></th>
<th>all 512-bit p</th>
<th>all 768-bit p</th>
<th>one 1024-bit p</th>
<th>ten 1024-bit p</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPS Top 1M MITM</td>
<td>45K (8.4%)</td>
<td>45K (8.4%)</td>
<td>205K (37.1%)</td>
<td>309K (56.1%)</td>
</tr>
<tr>
<td>HTTPS Top 1M</td>
<td>118 (0.0%)</td>
<td>407 (0.1%)</td>
<td>98.5K (17.9%)</td>
<td>132K (24.0%)</td>
</tr>
<tr>
<td>HTTPS Trusted MITM</td>
<td>489K (3.4%)</td>
<td>556K (3.9%)</td>
<td>1.84M (12.8%)</td>
<td>3.41M (23.8%)</td>
</tr>
<tr>
<td>HTTPS Trusted</td>
<td>1K (0.0%)</td>
<td>46.7K (0.3%)</td>
<td>939K (6.56%)</td>
<td>1.43M (10.0%)</td>
</tr>
<tr>
<td>IKEv1 IPv4</td>
<td>–</td>
<td>64K (2.6%)</td>
<td>1.69M (66.1%)</td>
<td>1.69M (66.1%)</td>
</tr>
<tr>
<td>IKEv2 IPv4</td>
<td>–</td>
<td>66K (5.8%)</td>
<td>726K (63.9%)</td>
<td>726K (63.9%)</td>
</tr>
<tr>
<td>SSH IPv4</td>
<td>–</td>
<td>–</td>
<td>3.6M (25.7%)</td>
<td>3.6M (25.7%)</td>
</tr>
</tbody>
</table>

Vulnerable servers, if the attacker can precompute for . . .
Practical Mitigations

- Move to elliptic curve cryptography
- If ECC isn’t an option, move to $\geq 2018$-bit primes.
- If 2048-bit primes aren’t an option, generate a fresh 1024-bit prime.
- Major browsers will refuse to accept 512-bit groups, will sunset 768 and 1024-bit soon.

Thoughts for theoreticians

- How do you future-proof new constructions?
- How do you protect against implementation mistakes?
- How do you defend against politics?

weakdh.org