Public Key Cryptography and Protocols

Simon Foley

October 27, 2015

Cryptographic Ciphers: Recap

A cryptographic cipher is a pair of Encrypt and Decrypt algorithms such that given *plaintext* P, encryption key K_1 and decryption key K_2 then

$$D(K_2, E(K_1, P)) = P$$

- In absence of knowledge about K_2, it must be not be feasible to recover P from the ciphertext $E(K_1, P)$.
- Given P and $E(K_1, P)$, it must not be feasible to recover K_1

Note that the *plaintext* P can be any data, including plain text.

We call $E(K_1, P)$ the *ciphertext*.

Symmetric Cryptography: $K1 = K2$. This may also be called *secret key cryptography*.

Asymmetric Cryptography: $K1 \neq K2$. This is commonly called *public-key cryptography*.

Modular Arithmetic: Exponentiation

- Calculating $2^{16} = 2 \times 2 \times \cdots \times 2$ requires 15 "$\times$" operations to raise 2 to the power of this 4-bit number. In general computing g^e in this way is bounded by e operations, i.e., is exponential in the *size* of e.
- Improve strategy by calculating 2^{16} in 4 operations:

  $$2^{16} = (2^8)^2 = ((2^4)^2)^2 = (((2^2)^2)^2)^2$$

  In general, computing g^e in this way needs at best $\mathcal{O}(log_2(e))$ operations and is linear in the *size* of e.
- Generalize strategy: (observe $x^{2n} = (x^n)^2$ and $x^{n+1} = x \times x^n$)

  $$\begin{aligned} 2^9 &= 2 \times 2^8 \\ &= 2 \times ((2^2)^2)^2 \end{aligned}$$

  $$\begin{aligned} 2^7 &= 2 \times 2^6 \\ &= 2 \times (2^3)^2 = 2 \times (2 \times 2^2)^2 \end{aligned}$$

Modular Arithmetic: Exponentiation

Calculate

$$5^{58} \bmod 97 = (\text{a really big 40 digit number}) \bmod 97 = 44$$

There's an easy way to manage such large numbers by using property:

$$(a \times b) \bmod c = ((a \bmod c) \times (b \bmod c)) \bmod c$$

Example

$$\begin{aligned} 5^{58} \bmod 97 &= (5^{29} \bmod 97)^2 \bmod 97 \\ &= ((5 \bmod 97) \times (5^{14} \bmod 97)^2 \bmod 97)^2 \bmod 97 \\ &= \cdots \\ &= (5 \times 48^2 \bmod 97)^2 \bmod 97 \\ &= 44 \bmod 97 \end{aligned}$$

Modular Arithmetic: Exponentiation Algorithm

From the observations on the previous two slides we can conclude that it is computationally *feasible* to compute $g^x \bmod n$ for large x, n.

Algorithm fastExp(g,e,n) implements $g^e \bmod n$:

```
fastExp(g,0,n)  = 1
fastExp(g,e,n)  = if odd(e) then
                    return (g * fastExp(g, e-1, n) mod n)
                  else
                    return (sqr(fastExp(g, e/2, n) mod n))
```

Modular Arithmetic: Discrete Logarithm

- Let g be a *primitive root* of n, i.e., the powers (modulo n) of g generate all integers from 1 to $n-1$: $g^1 \bmod n, g^2 \bmod n, \ldots, g^{n-1} \bmod n$ are distinct, consisting of integers 1 thru $n-1$ in some permutation.
- n is a prime number
- For any integer x, $0 < x < n$ and primitive root g of n one can find a unique exponent e such that the equation $x = g^e \bmod n$ holds.
- e is the *"discrete logarithm"* of x (generator g, modulus n).
- For example, the discrete log of 44 (generator 5, modulus 97) is 58.
- Given g, n then computing e from $x = g^e \bmod n$ is hard since, we intuitively have to 'test' the equation for every $e = 1, 2, 3, \ldots n-1$ (for large n).
- Computing the discrete logarithm is exponential in the size of n and is not computationally feasible to compute if n is large.

Diffie Hellman Key Exchange

Alice and Bob agree on an encryption key over a network. Steps:

- ☐ Alice and Bob agree on a good g and n. Can be done in public.
- ☐ Alice picks a large random integer x and computes $X = g^x \bmod n$. She keeps x secret, but it doesn't matter who knows X (discrete log).
- ☐ Bob behaves in the same way, picking y and computing $Y = g^y \bmod n$.
- ☐ Alice sends X to Bob, and Bob sends Y to Alice.
- ☐ Alice computes $k = Y^x \bmod n$ and Bob computes $k' = X^y \bmod n$.
- ☐ By modular arithmetic $k = g^{x*y} \bmod n = k'$. k is secret key between Alice and Bob. No one listening on the channel can compute key g^{x*y} in a reasonable amount of time (calculate discrete log of X and Y).
- ☐ Be careful picking g, n. `http://tools.ietf.org/html/rfc5114`

How Diffie Hellman Fails in practice [2015]

Many implementations of DH use standard 'safe' generator g and modulus n parameters; these are specially selected to ensure effectiveness (no short-cuts possible when computing discrete log).

Source	Popularity	Prime
Apache	82%	`9fdb8b8a004544f0045f1737d0ba2e0b 274cdf1a9f588218fb435316a16e3741 71fd19d8d8f37c39bf863fd60e3e3006 80a3030c6e4c3757d08f70e6aa871033`
mod_ssl	10%	`d4bcd52406f69b35994b88de5db89682 c8157f62d8f33633ee5772f11f05ab22 d6b5145b9f241e5acc31ff090a4bc711 48976f76795094e71e7903529f5a824b`
(*others*)	8%	(463 distinct primes)

Table 1: **Top 512-bit DH primes for TLS.** 8.4% of Alexa Top 1M HTTPS domains allow DHE_EXPORT, of which 92.3% use one of the two most popular primes, shown here.

How Diffie Hellman Fails in practice [2015]

Many implementations of DH use standard 'safe' generator g and modulus n parameters; these are specially selected to ensure effectiveness (no short-cuts possible when computing discrete log).

Source	Popularity	Prime

Apache	82%	`9fdb8b8a004544f0045f1737d0ba2e0b`
		`274cdf1a9f588218fb435316a16e3741`
		`71fd19d8d8f37c39bf863fd60e3e3006`
		`80a3030c6e4c3757d08f70e6aa871033`

mod_ssl	10%	`d4bcd52406f69b35994b88de5db89682`
		`c8157f62d8f33633ee5772f11f05ab22`
		`d6b5145b9f241e5acc31ff090a4bc711`
		`48976f76795094e71e7903529f5a824b`

(*others*)	8%	(463 distinct primes)

Table 1: **Top 512-bit DH primes for TLS.** 8.4% of Alexa Top 1M HTTPS domains allow DHE_EXPORT, of which 92.3% use one of the two most popular primes, shown here.

With advances in computing discrete logs its feasible to carry out a *precomputation* (10 core years) attack that involves building a table (2.5GB) for a 512-bit modulus that then makes computing any discrete log (that uses the modulus) feasible (10 mins).

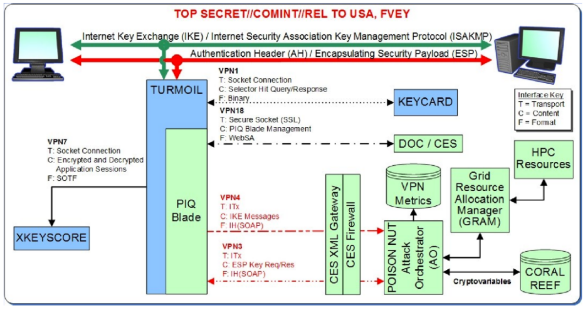


Figure 4: **NSA's VPN decryption infrastructure.** This classified illustration published by Der Spiegel [67] shows captured IKE handshake messages being passed to a high-performance computing system, which returns the symmetric keys for ESP session traffic. The details of this attack are consistent with an efficient break for 1024-bit Diffie-Hellman.

Snowden documents suggest that NSA may be exploiting this for 1024 bit modulus. Recent estimates argue a 45M core-years precomputation at a cost of cost $11B (1-calendar year) for a given modulus.

See *Weak Diffie-Hellman and the Logjam Attack* [`https://weakdh.org/`]
`https://www.eff.org/deeplinks/2015/10/how-to-protect-yourself-from-nsa-attacks-1024-bit-DH`

Diffie Hellman Key Exchange: Bucket Brigade Attack

Basic DH Protocol (in any order):

$$\begin{aligned} \text{Msg1} \quad & A \to B: \quad g^x \bmod n \\ \text{Msg2} \quad & B \to A: \quad g^y \bmod n \end{aligned}$$

Neither party really knows with whom it shares secret $k = g^{xy} \bmod n$.

Eve routes messages between Alice and Bob

$$\begin{aligned} \text{Msg}\alpha 1 \quad & A \to B[E]: \quad g^x \bmod n \\ \text{Msg}\beta 1 \quad & A[E] \to B: \quad g^z \bmod n \\ \text{Msg}\alpha 2 \quad & B[E] \to A: \quad g^z \bmod n \\ \text{Msg}\beta 2 \quad & B \to A[E]: \quad g^y \bmod n \end{aligned}$$

Alice uses $k_a = g^{xz} \bmod n$ to 'speak' with Bob, Bob uses $k_b = g^{zy} \bmod n$ to 'speak' with Alice, and Eve knows both keys, while carrying out a man-in-the-middle attack.

Diffie Hellman Key Exchange

The basic DH protocol does not provide authentication.

We need to be careful how we extend the protocol to include authentication. For example, simply adding passwords as follows does not work:

Msg1 $A \rightarrow B:$ g^x mod n

Msg2 $B \rightarrow A:$ g^y mod n

Msg3 $A \rightarrow B:$ $\{Alice's\ password\}_{g^x}$ mod n

Msg3 $B \rightarrow A:$ $\{Bob's\ password\}_{g^x}$ mod n

In this case the attacker can still carry out the bucket brigade/man in the middle attack.

DH Exchange: achieving Authentication

Think of $X = g^x \bmod n$ as something that Alice makes public, while x is something she keeps private.

If Bob can be sure that the public $X = g^x \bmod n$ was generated/owned by Alice and Alice can be sure that the public $Y = g^y \bmod n$ was generated/owned by Bob, then we do have an authentic connection between Alice and Bob. In this case, the DH protocol can be used to establish a shared secret key.

$$\begin{array}{lll} \text{Msg1} & A \to B: & g^x \bmod n \\ \text{Msg2} & B \to A: & g^y \bmod n \end{array}$$

However, publicly stating 'Alice=X' on the network is not sufficient.

DH is a very common protocol for exchanging keys when the authenticity (owners) of X and Y are known;

Other protocols are used to establish authenticity of X and Y.

Public Key Cryptography Overview

Given plaintext P, ciphertext C, key K and its inverse key K^{-1}:

$$C = \{P\}_K \qquad P = \{C\}_{K^{-1}}$$

Intuitively, think of K as encryption key and K^{-1} as decryption key.

In absence of knowledge about K, not feasible to recover P from C.
Not feasible to recover K from P and/or C.
Not feasible to determine K^{-1} from K.

Alice keeps K_A^{-1} secret ("private key"), publishes K_A ("public key"). Bob does same for different K_B, K_B^{-1}.

Confidentiality: Alice sends $\{M\}_{K_B}$ to Bob, trusting that Bob does not share K_B^{-1} with anyone.

Authentication: Alice *signs* message M as $\{M\}_{K_A^{-1}}$. Anyone can confirm her signature by decrypting with her public key K_A.

Combined: $Alice \rightarrow Bob : \{\{M\}_{K_A^{-1}}\}_{K_B}$. A signed letter in an envelope!

RSA Public Key Cryptography (Sketch)

Choose two large prime numbers p and q and let $n = p \times q$.

The "totient" $\phi(n)$ of n is the number of numbers less than n with no factors in common with n.

Pick integer $e < n$ relatively prime to $\phi(n)$.
Find a second integer d such that $e \times d \bmod n = 1$.
It turns out that knowing $\phi(n)$ makes it easy/feasible to find this d.

The public key is (e, n), the private key is (d, n).

Let $m < n$ represent a message, then

- $c = m^e \bmod n$ is the encryption of m with the public key, and
- $m = c^d \bmod n$ is the decryption of m with the private key.

Break an arbitrary length message into $\lceil log_2(n) \rceil$ sized-blocks and encrypt a block at a time ($\lceil log_2(n) \rceil$ gives number of bits needed to represent n).

The size of the key in RSA is typically considered to be the size of the modulus.

RSA Example

Pick $p = 47$; $q = 71$; $n = pq = 3337$.

$\phi(n) = (p-1)(q-1) = 3220$.

Randomly pick $e = 79$ such that $gcd(e, \phi(n)) = 1$

Compute $d = e^{-1} [\text{mod}\, \phi(n)] = fastexp(79, 3320 - 1, 3220) = 1019$.

To encrypt message 688232686, break into blocks $688\ 232\ 686$.

b	$c = b^e \bmod n$	$c^d \bmod n$
688	$688^{79} \bmod 3337 = 1570$	$1570^{1019} \bmod 3337 = 688$
232	$232^{79} \bmod 3337 = 2756$	$2756^{1019} \bmod 3337 = 232$
686	$686^{79} \bmod 3337 = 2091$	$2091^{1019} \bmod 3337 = 686$

Note this example does not consider the problem of traffic analysis: an attacker can detect patterns in the ciphertext. How would we prevent this?

Note: in practice we don't use RSA to encrypt bulk data.

RSA Public Key Scheme

We do not consider the mathematics behind RSA. However, there are a number of important requirements that have to be observed.

- Given the public key (e, n), then the decryption key can be computed if the primes p and q are publicly known. Therefore, the primes p and q must not be revealed by the principal that generates the public/private key pair.
- The primes p and q have to be large enough to ensure that it is not possible to factor them from the public modulus n.
- In practice, a principal generates random p and q and computes (e, n) and (d, n) and then discards p and q. The principal keeps (d, n) private and makes key (e, n) public.
- In practice, primes should be at least 2048 bits long, and be 'suitable'.
- Many other practical requirements on the properties of the numbers.

Use RSA implementations that are compliant with standards!

Comparable Key Strengths

From NIST Special Publication 800-57, 2015, *Recommendation for Key Management - Part 1: General (Revision 4)*.

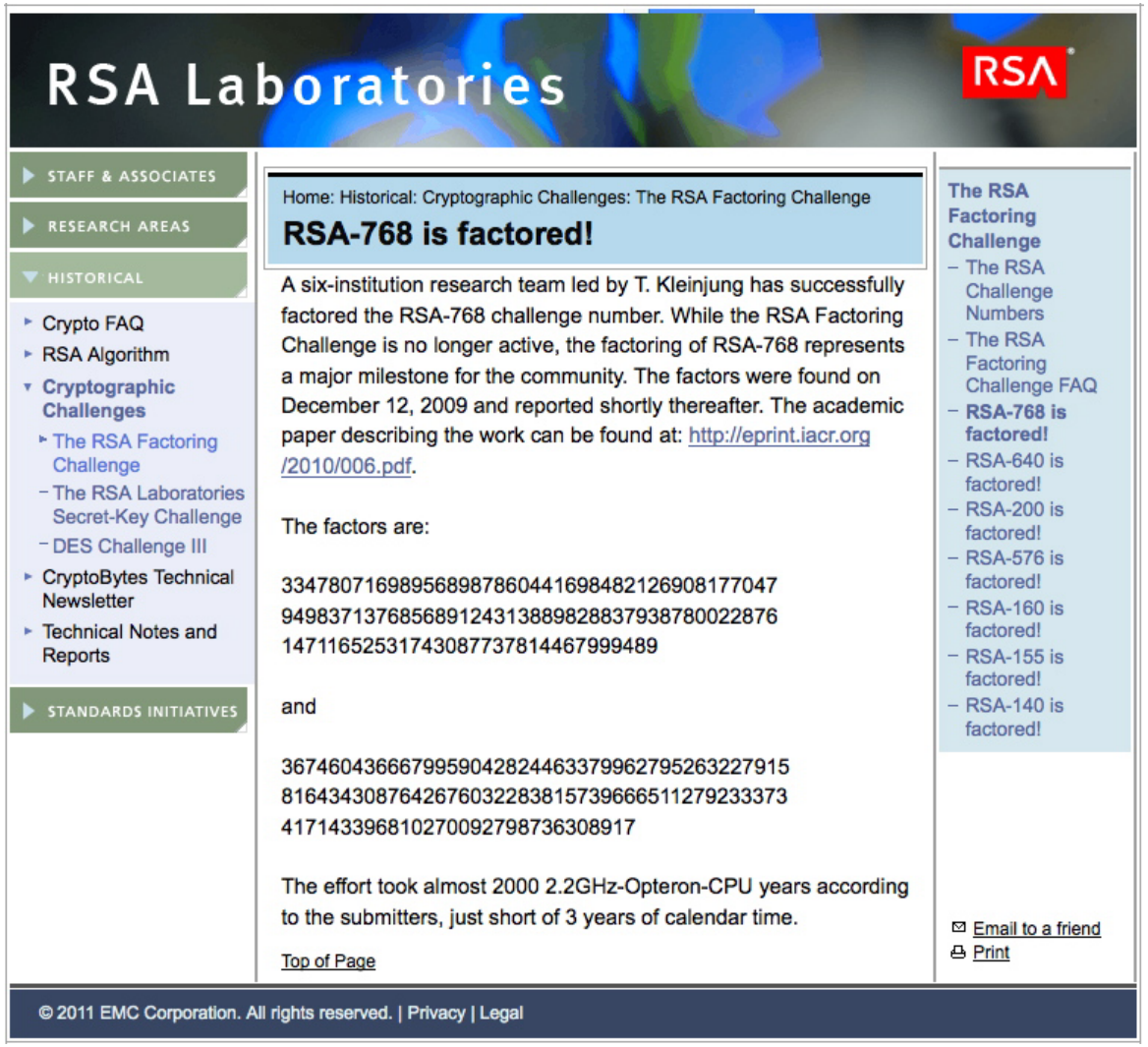
Table 2: Comparable strengths

Security Strength	Symmetric key algorithms	FFC (e.g., DSA, D-H)	IFC (e.g., RSA)	ECC (e.g., ECDSA)
≤ 80	2TDEA[23]	$L = 1024$ $N = 160$	$k = 1024$	$f = 160$-223
112	3TDEA	$L = 2048$ $N = 224$	$k = 2048$	$f = 224$-255
128	AES-128	$L = 3072$ $N = 256$	$k = 3072$	$f = 256$-383
192	AES-192	$L = 7680$ $N = 384$	$k = 7680$	$f = 384$-511
256	AES-256	$L = 15360$ $N = 512$	$k = 15360$	$f = 512+$

`http://csrc.nist.gov/publications/drafts/800-57/sp800-57p1r4_draft.pdf`

RSA Factoring Challenge: remember the public key is public

RSA Factoring Challenge



ars technica

Main Menu ▾ My Stories: ▾ Forums Subscribe Jobs

RISK ASSESSMENT / SECUF
HACKTIVISM

Breaking 512-bit RSA with Amazon EC2 is a cinch. So why all the weak keys?

"Factorization as a service" in Amazon cloud is so easy novices can do it.

by **Dan Goodin** - Oct 20, 2015 1:00 pm UTC

Share Tweet 31

martinak15

The cost and time required to break 512-bit RSA encryption keys has plummeted to an all-time low of just $75 and four hours using a recently published recipe that even computing novices can follow. But despite the ease and low cost, reliance on the weak keys to secure e-mails, secure-shell transactions, and other sensitive communications remains alarmingly high.

http://arstechnica.com/security/2015/10/breaking-512-bit-rsa-with-amazon-ec2-is-a-cinch-so-why-all-the-weak-keys/
https://eprint.iacr.org/2015/1000.pdf

Recommended Key Sizes

From NIST Special Publication 800-57, 2015, *Recommendation for Key Management - Part 3: Application-specific key management guide (Revision 1)*.

Table 2-1: Recommended Algorithms and Key Sizes

Key Type	Algorithms and Key Sizes
Digital Signature keys used for authentication (for Users or Devices)	RSA (2048 bits) ECDSA (Curve P-256)
Digital Signature keys used for non-repudiation (for Users or Devices)	RSA (2048 bits) ECDSA (Curves P-256 or P-384)
CA and OCSP Responder Signing Keys	RSA (2048 or 3072bits) ECDSA (Curves P-256 or P-384)
Key Establishment keys (for Users or Devices)	RSA (2048 bits) Diffie-Hellman (2048 bits) ECDH (Curves P-256 or P-384)

`http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf`

"FREAK" flaw in Android and Apple devices cripples HTTPS crypto protection

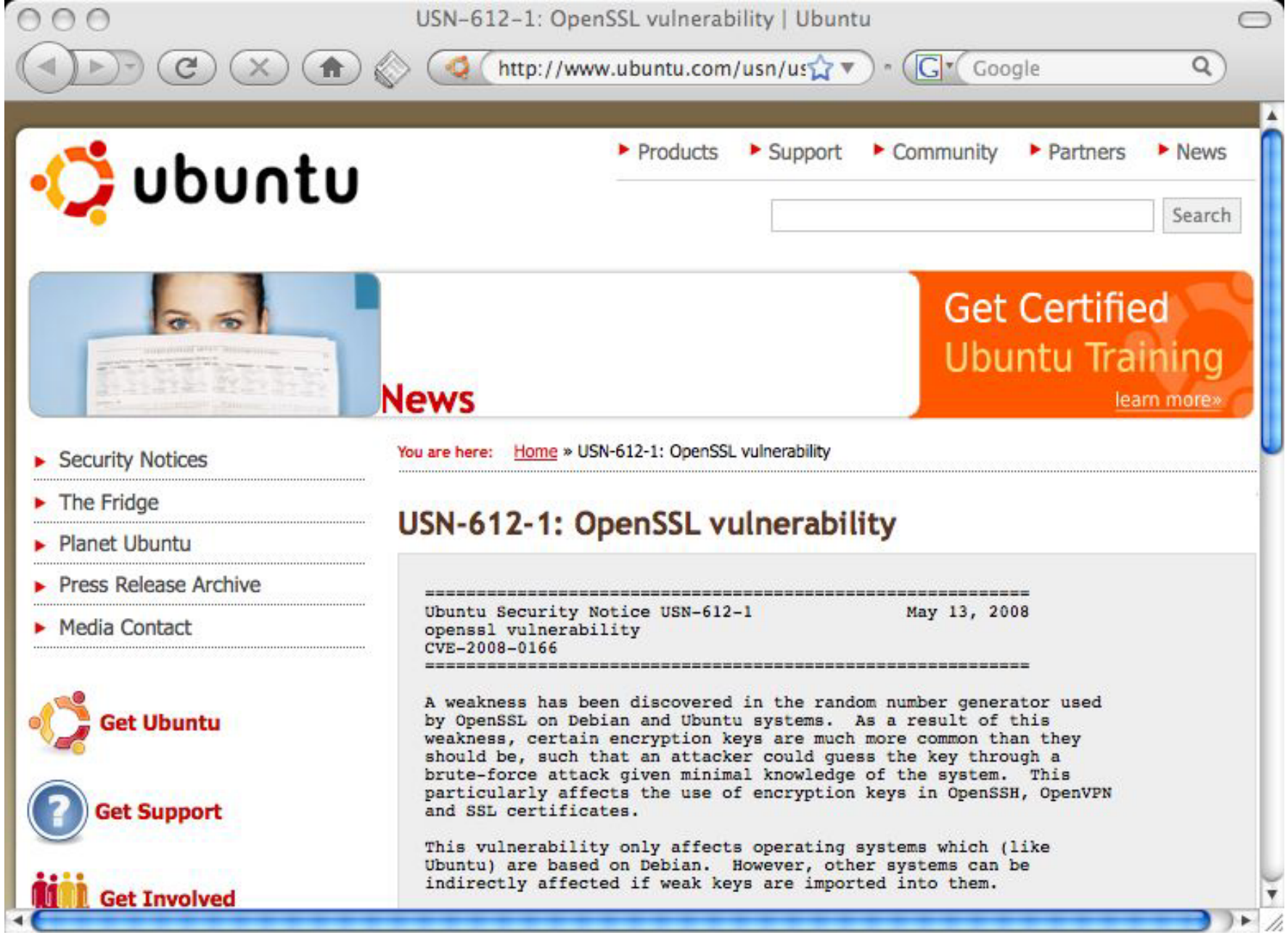
Bug forces millions of sites to use easily breakable key once thought to be dead.

by **Dan Goodin** - Mar 3, 2015 9:07 pm UTC

Aurich Lawson / Thinkstock

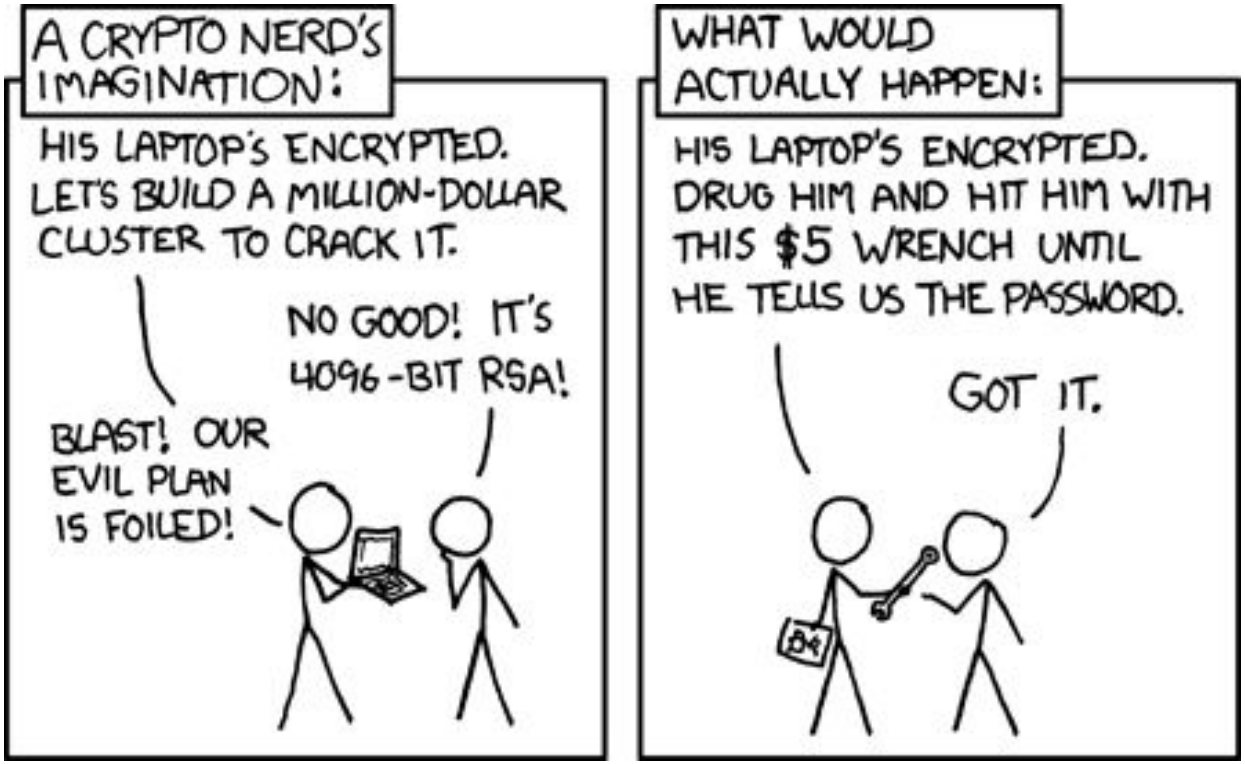
Security experts have discovered a potentially catastrophic flaw that for more than a decade has made it possible for attackers to decrypt HTTPS-protected traffic passing between Android or Apple devices and hundreds of thousands or millions of websites, including AmericanExpress.com, Bloomberg.com, NSA.gov, and FBI.gov.

http://arstechnica.com/security/2015/10/breaking-512-bit-rsa-with-amazon-ec2-is-a-cinch-so-why-all-the-weak-keys/



Developer accidentally removed lines that referenced un-initialized memory that provided entropy for random number generation used to generate primes p and q. This makes it possible to brute force factor public modulus to determine p and q and calculate the private key.

Rubber Hose Cryptography [xkcd.com]



Example: Public Key Cryptography Standards PKCS1/RFC3447

PKCS#1 . . . PKCS#15 are a set of standards that specify safe ways of encoding encrypted material. They are designed to avoid known threats: one does not have to be a cryptographer to use RSA safely.

For example: we want to avoid pasword guessing; encrypted material can be vulnerable if it is significantly shorter than public modulus (eg, a secret key). PKCS1 requires that the message include at least 8 bytes of random data.

RFC3447 publishes full details of PKCS1
[`http://www.ietf.org/rfc/rfc3447.txt`]

For simplicity, we let $\{P\}_K$ denote a suitable implementation of the RSA function applied to data P using key K. We denote the inverse of key K as K^{-1} (and the inverse of K^{-1} is K). Thus $P = \{\{P\}_K\}_{K^{-1}}$.

Since the RSA encryption and decryption operations are identical we don't make any special distinction between K and K^{-1} and thus we also have that $D = \{\{D\}_K^{-1}\}_K$

Public Key Attack (Forward Search)

Suppose that Alice and Bob follow the protocol:

$$Alice \to Bob : \{\{M\}_{K_A^{-1}}\}_{K_B}$$

Eve tests lots of random values R_j until $\{R_j\}_{K_A}$ looks like a valid message.

$$\{R_j\}_{K_A} = \texttt{buy shares}$$

Eve, pretending to be Alice, then sends $M = \{R_j\}_{K_B}$ to Bob.

Bob, thinking that the message originated from Alice computes

$$\{M\}_{K_B^{-1}} = R_j$$

and then to check signature from A

$$\{R_j\}_{K_A} = \texttt{buy shares}$$

Solution: Add redundancy to message

$$Alice \to Bob : \{\{M, Alice\}_{K_A^{-1}}\}_{K_B}$$

Public Key Cryptography: Encrypting Bulk Data

Public key algorithms involve computationally intensive calculations and, therefore, its not effective to encrypt entire message using public key cipher.

In practice, a symmetric cipher should be used to perform bulk encryption, while the public key cipher is used to secure the symmetric session key. In SW DES about 100 times faster than RSA; difference even greater in HW.

For example, assuming that Bob knows Alice's public key K_A, then, Bob proposes that Alice should use symmetric key K_{AB}:

$$\begin{aligned} \text{Msg1}: \quad & B \rightarrow A: \quad \{K_{AB}, N_A\}_{K_A} \\ \text{Msg2}: \quad & A \rightarrow B: \quad \{N_A, Msg\}_{K_{AB}} \end{aligned}$$

Alice uses her private key K_A^{-1} to setup session key K_{AB}.

This is not entirely effective, there is authentication of Alice but Bob is not authenticated: Bob knows that he shares a key with Alice, not vice-versa.

This one-sided authentication is not unlike the secure browsing relationship between a web-browser (Bob) and website (Alice).

Example: A Public Key Protocol with a flaw

There are public-key versions of Needham-Schroeder/Kerberos and we will study SSL later. To help understand the requirements, the following is a sample protocol with a flaw.

Goal is to provide secure and authentic key exchange. Assume everyone knows the public key of everyone else. Use public key encryption to exchange (symmetric) keys; T_A a timestamp for freshness.

$$\text{Msg } 1 \quad A \to B : \quad \{\{T_A, K_{AB}, A\}_{K_A^{-1}}\}_{K_B}$$

In receiving $\{T_A, K_{AB}, A\}_{K_A^{-1}}$, Bob believes that the key K_{AB} could only have been proposed by/come from Alice.

By encrypting K_{AB} with Bob's public key in $\{\{T_A, K_{AB}, A\}_{K_A^{-1}}\}_{K_B}$, Alice believes that the only person able to discover K_{AB} will be Bob.

Therefore, if Alice subsequently receives a message encrypted under symmetric key K_{AB} then it seems reasonable for her to believe it came from Bob, and vice-versa. However, ...

Example: A Public Key Protocol with a flaw

Consider the protocol:

$$\text{Msg } 1 \quad A \to B: \quad \{\{T_A, K_{AB}, A\}_{K_A^{-1}}\}_{K_B}$$

At some point Alice contacts Eve:

$$\text{Msg } \alpha 1 \quad A \to E: \quad \{\{T_A, K_{AE}, A\}_{K_A^{-1}}\}_{K_E}$$

Eve decrypts message to give $X = \{T_A, K_{AE}, A\}_{K_A^{-1}}$.
Eve encrypts X with K_B and sends

$$\text{Msg } \gamma 1 \quad A[E] \to B: \quad \{\{T_A, K_{AE}, A\}_{K_A^{-1}}\}_{K_B}$$

Eve can now masquerade as Alice to Bob! Fix it!

RSA Digital Signatures

Principal A (customer) generates RSA key pair K_A^{-1}, K_A.

A signs a message M (eg, an order) $RSA(K_A^{-1}, M)$.

In practice, A signs a digest, eg, $RSA(K_A^{-1} SHA2(M))$.

Send signed message to Principal B (Merchant)

$$A \rightarrow B : M, \boxed{RSA(K_A^{-1}, SHA1(M))}$$

Principal B uses A's public key to confirm signature.

$$SHA2(M) = RSA(K_A, \boxed{RSA(K_A^{-1}, SHA2(M))})$$

Other public-key schemes can be used to provide digital signatures, for example, the Digital Signature Standard is based on a specialised public key cipher designed specifically for signatures.

We often use the notation $\{M\}_{sK_A}$ to denote the message M signed by the owner of public key K_A.

RSA Digital Signatures

Digital signatures sign a cryptographic one-way hash of the message/document.

We want to be sure that it is not possible for two different documents to have the same hash value (and therefore the same signature). If it was possible for two different documents M, M' to have the same hash value $h(M) = h(M')$ then a malicious user might trick a victim into accepting document M with its signature $RSA(K_A^{-1}h(M))$ as valid, while at a later date claiming that the document was actually M' with its same signature $RSA(K_A^{-1}h(M'))$.

Remember the property of a one-way hash function.

A function h maps arbitrary length value x to fixed length value y such that:

- *Collision freeness. Hard to find values x, x' such that $h(x) = h(x')$.*

Therefore, we avoid hash functions that cannot provide strong assurances of collision freedom. For example, MD5 and SHA1.

Postscript files with same signature/MD5 hash value a25f7f0b 29ee0b39 68c86073 8533a4b9

Julius. Caesar
Via Appia 1
Rome, The Roman Empire

May, 22, 2005

To Whom it May Concern:

Alice Falbala fulfilled all the requirements of the Roman Empire intern position. She was excellent at translating roman into her gaul native language, learned very rapidly, and worked with considerable independence and confidence.

Her basic work habits such as punctuality, interpersonal deportment, communication skills, and completing assigned and self-determined goals were all excellent.

I recommend Alice for challenging positions in which creativity, reliability, and language skills are required.

I highly recommend hiring her. If you'd like to discuss her attributes in more detail, please don't hesitate to contact me.

Sincerely,

Julius Caesar

Julius. Caesar
Via Appia 1
Rome, The Roman Empire

May, 22, 2005

Order:

Alice Falbala is given full access to all confidential and secret information about GAUL.

Sincerely,

Julius Caesar

[from: Lucks and Daum, *The Story of Alice and her Boss: Hash Functions and the Blind Passenger Attack*, Eurocrypt 2005 rump session.]

Recommended Key Sizes

From NIST Special Publication 800-57, 2008, *Recommendation for Key Management - Part 3: Application-specific key management guide (draft)*.

Table 2-2 Recommended Signature Algorithms

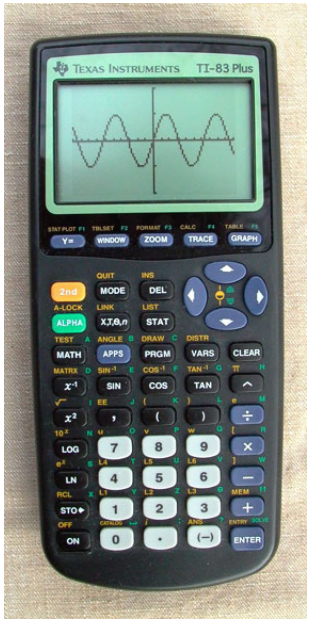
Signature Generation Date	Public Key Algorithms and Key Sizes	Hash Algorithms	Padding Scheme
Through 12/31/2009	RSA (2048, 3072, or 4096 bits)	SHA-1	PKCS #1 v1.5
	RSA (2048, 3072, or 4096 bits)	SHA-256	PKCS #1 v1.5
	ECDSA (Curve P-256)	SHA-256	N/A
	ECDSA (Curve P-384)	SHA-384	N/A
1/1/2010 through 12/31/2010	RSA (2048, 3072, or 4096 bits)	SHA-1	PKCS #1 v1.5
		SHA-256	PKCS #1 v1.5, PSS
	ECDSA (Curve P-256)	SHA-256	N/A
	ECDSA (Curve P-384)	SHA-384	N/A
After 12/31/2010	RSA (2048, 3072, or 4096 bits)	SHA-256	PKCS #1 v1.5, PSS
	ECDSA (Curve P-256)	SHA-256	N/A
	ECDSA (Curve P-384)	SHA-384	N/A

512 bit TI signing key factored [2009]

Updates for Texas Instruments programmable calculators must be signed by a (512 bit) private RSA key known only by TI. The calculator stores a (tamper-proof) copy of TI's public key and uses it to check the signature on updates. If not signed correctly then the calculator will not accept the update.

In 2009 an enthusiast factored the modulus in the public key and used the results to compute the private key. This was done by a brute-force search, taking several months on a fairly standard desktop PC using off the shelf factoring software.

The key was published on the enthusiast's website. TI's response was to have it lawyers send a DMCA take-down to the enthusiast, resulting in the Streisand effect (key subsequently published in a number of other places).



Who Generates the Public/Private Keys?

We call the principal who knows the private key K_A^{-1} (and the public key) the *owner* of the public key K_A.

In principle, the intended owner of public key K_A should be the principal that generates the public/private key pair (K_A^{-1}, K_A).

- If a third party generated the key pair on behalf of A, then principal A would have to trust the third party not to reveal the private key to anyone else.
- We would like to be able to believe that when we confirm that $\{M\}_{sK_A}$ was signed by the owner of K_A then the principal cannot easily repudiate their signature. This is subject to them not having declared their private key to be compromised.

If no ambiguity can arise then $\{M\}_{sK_A}$ is used to denote the signed message (by the owner of private key K_A). For example, $M, RSA(K_A^{-1}, SHA1(M))$.

Digital Signatures versus MAC

Customer and Merchant share secret key K for keyed hash function $h_K(M)$

$$\text{Customer} \rightarrow \text{Merchant} : order, h_K(order)$$

Customer and merchant can detect modifications by attacker.

MAC does not constitute evidence that a third party could use to decide whether customer or merchant generated message.

How can customer prove to judge that she ordered only 5 widgets when the merchant claims that it was 10, and both have MACs to (supposedly) prove it? MAC provides integrity check, not authentication check.

Digital Signature Scheme: A third party can resolve disputes about the validity of a digital signature without having to know the signer's key. Digital signature schemes should ideally support non-repudiation.

Be Careful What you Sign

Suppose that Alice and Bob know each others public key and want to exchange signed messages in secret. Need a combination of public key cryptography (for signing) and symmetric key cryptography (for secrey, etc).

Alice generates a symmetric session key K_{AB} which is used to encrypt the bulk data. Two strategies:

$$A \to B : \{M\}_{K_{AB}}, \{\{h(M), K_{AB}, A, B, \ldots\}_{K_A^{-1}}\}_{K_B} \quad (1)$$

$$A \to B : \{M, h(M)\}_{K_{AB}}, \{\{K_{AB}, A, B, \ldots\}_{K_A^{-1}}\}_{K_B} \quad (2)$$

Which protocol provides an effective signature?

Some Public Key Protocols

ISO/IEC 9798-3 Signature Key Three Pass MA Protocol. Alice and Bob know each other's public keys K_A and K_B, respectively and run the following protocol to check each other's presence.

Msg1	$A \to B$	A, N_a
Msg2	$B \to A$	$\{A, N_a, N_b\}_{sKb}$
Msg3	$A \to B$	$\{N_a, N_b\}_{sKa}$

This protocol supports optional attributes in the messages and can be used to, for example, carry out an authentic DH key exchange.

Msg1	$A \to B$	A, N_a
Msg2	$B \to A$	$\{A, N_a, N_b, g^y \bmod n\}_{sKb}$
Msg3	$A \to B$	$\{N_a, N_b, g^x \bmod n\}_{sKa}$

where g and n give DH generator and modulus, respectively. x and y are the secret exponents of A and B. On completion, A holds authenticated $g^y \bmod n$ from B, and similarly for B; both share secret $g^{xy} \bmod n$.

Some Public Key Protocols: A Preliminary Sketch of SSL

Alice uses an SSL-style protocol to connect to the web-server Bob.

Msg1	$A \to B$	$A, \{A, N_A, K_{AB}, \ldots\}_{K_B}$
Msg2	$B \to A$	$\{B, N_A, N_B \ldots\}_{K_{AB}}$
Msg3	$A \to B$	$\{A, N_B + 1\}_{K_{AB}}$
...	$A \leftrightarrow B$	$\{data, etc\}_{K_{AB}}$

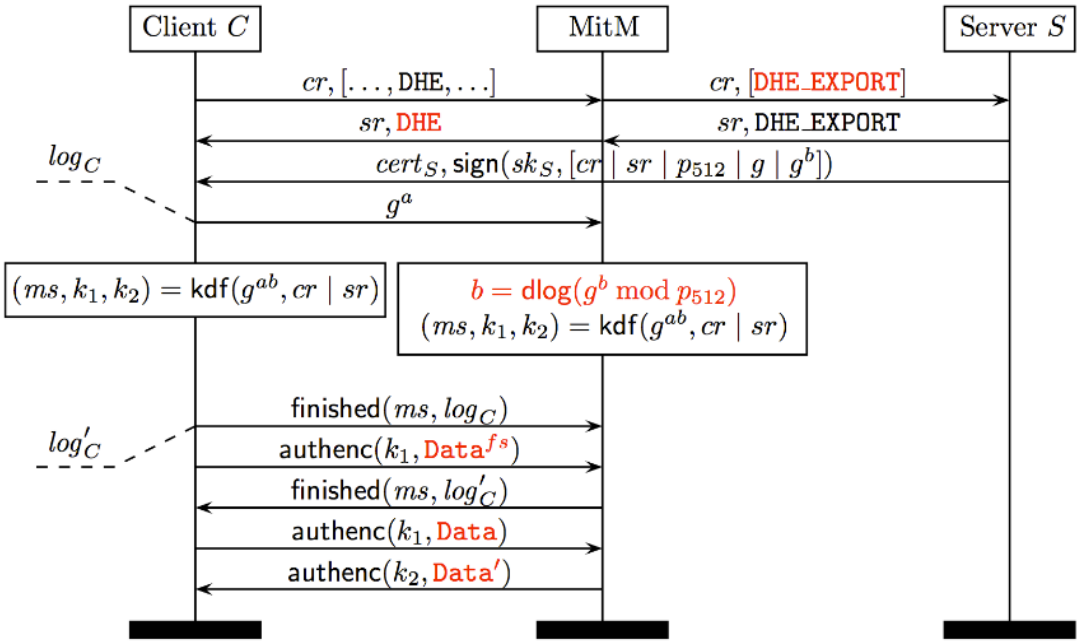
In this protocol, we assume that Alice knows the Server Bob's public key K_B, which is used to establish a secret shared session key K_{AB}.

This protocol provides authentication of Bob. Alice knows for sure that when she encrypts her messages with K_{AB} then only Bob can read them, since only Bob can decrypt message 1 to discover K_{AB}.

The protocol does not provide authentication of Alice. Bob does not know the initiator of the protocol. If (weak) authentication of the client is required then, once connected, Alice could provide a password/pin at a login web-page. Note that in general SSL can also provide client-side authentication as part of the protocol (we'll see this later).

Logjam attack on SSL

Allows a man-in-the-middle attacker to downgrade vulnerable TLS connections to a 512-bit export-grade DH exchanged key.



Disable support for Export grade crypto in browser!

`http://blog.cryptographyengineering.com/2015/05/attack-of-week-logjam.html`

Station to Station Protocol [Diffie VanOorschot Wiener]

A Diffie-Hellman key exchange that includes an exchange of authentication signatures.

$$
\begin{array}{lll}
\text{Msg1} & A \to B & g^x \bmod p \\
\text{Msg2} & B \to A & g^y \bmod p, \{\{g^x \bmod p, g^y \bmod p\}_{sK_B}\}_K \\
\text{Msg3} & A \to B & \{\{g^x \bmod p, g^y \bmod p\}_{sK_A}\}_K
\end{array}
$$

where Alice and Bob own public keys K_A and K_B, respectively, and are somehow known to each other.

The generator g and modulus p are publicly known/exchanged. Alice and Bob generate local secrets x and y, respectively, and the shared DH secret is $K = g^{xy} \bmod p$.

Features of Station to Station Protocol

Perfect Forward Secrecy Disclosure of long-term secret keying material does not compromise secrecy of exchanged keys from earlier runs[1].

Direct Authentication Authentication is established by the end of the protocol run. Both parties have proven that they know the secret key.

No Timestamps

[1] Unlike, the protocol $A \rightarrow B : \{\{A, B, K\}_{K_A^{-1}}\}_{K_B}$ where an attacker that discovers K_B^{-1} can extract earlier keys from earlier encrypted exchanges.

Station to Station Protocol in Practice

Msg1	$A \rightarrow B$	$g, p, , g^x \bmod p$
Msg2	$B \rightarrow A$	$Cert_{K_B} g^y \bmod p, \{\{g^x \bmod p, g^y \bmod p\}_{sK_B}\}_K$
Msg3	$B \rightarrow A$	$Cert_{K_A}, \{\{g^x \bmod p, g^y \bmod p\}_{sK_A}\}_K$

where $Cert_{K_A} = \{Alice, K_A, g, p\}_{sK_T}$, where K_T is a trusted third party. and g, p must be signed.

Can run an authentication-only version of STS,

Msg1	$A \rightarrow B$	R_A
Msg2	$B \rightarrow A$	$Cert_{K_B}, R_B, \{R_B, R_A\}_{sK_B}$
Msg3	$A \rightarrow B$	$Cert_{K_A}, \{R_A, R_B\}_{sK_A}$

which is essentially ISO/IEC 9798 Three Message MA protocol (which includes optional fields for symmetric key to be exchanged).