1720 - Forward Secrecy: How to Secure SSL from Attacks by Government Agencies

Dave Corbett

Technical Product Manager

Agenda

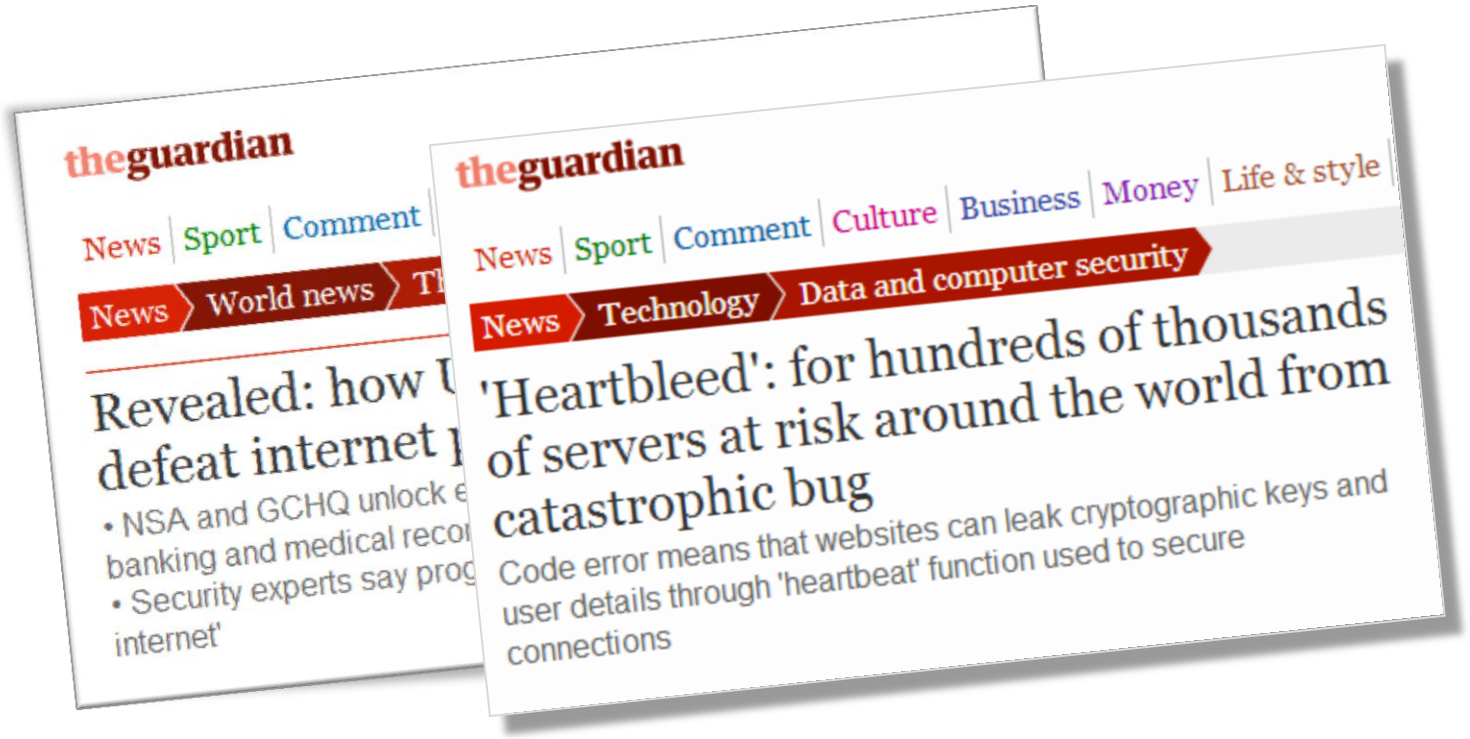
Part 1: Introduction

- Why is Forward Secrecy important?
- Technology Overview
  - RSA Algorithm
  - Diffie-Hellman
- Performance Impact
- Increasing adoption of Forward Secrecy

Part 2: Deep Dive

- RSA Algorithm
- Diffie-Hellman
- Elliptic Curve Cryptography (ECC)
- Implementing Forward Secrecy

What's the problem?



http://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security
http://www.theguardian.com/technology/2014/apr/08/heartbleed-bug-puts-encryption-at-risk-for-hundreds-of-thousands-of-servers

Why is Forward Secrecy important?

- Using typical HTTPS implementations based on RSA, an adversary could record encrypted data and decrypt it later if they gain access to the relevant private key(s)
- Using Forward Secrecy this approach is impossible

http://blogs.computerworld.com/encryption/22366/can-nsa-see-through-encrypted-web-pages-maybe-so

Review of SSL/TLS

FUNCTION	Examples
Authentication	RSA DSA
Key Exchange/Agreement	RSA Diffie-Hellman
Encryption Algorithm	RC4 AES
Hashing Algorithm	MD5 SHA-1

RSA is the problem…

Ron Rivest, Adi Shamir, and Len Adleman released the Rivest-Shamir-Adleman (RSA) public key algorithm in 1978.

This algorithm can be used for encrypting and signing data.

The encryption and signing processes are performed through a series of modular multiplications.

A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

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

SSL Handshake Process
RSA

Browser requests web server identity

Client Hello
Challenge string
Supported cipher suites

Server Hello
Certificate (includes public key)
Confirmed cipher suite

sends encrypted session key

Master Key
Encrypted with server public key

Server sends an acknowledgement

Server Verify
Encrypted challenge string

Encrypted Data

RSA key exchange

- During the SSL handshake, the client receives the server's **certificate**
- This includes the server's **public key**
- Using **RSA**, the HTTPS **session key** is encrypted by the client using the server's **public key** and sent to the server across the network
- The server decrypts the session key using the corresponding **private key**

RSA is vulnerable if the private key is discovered

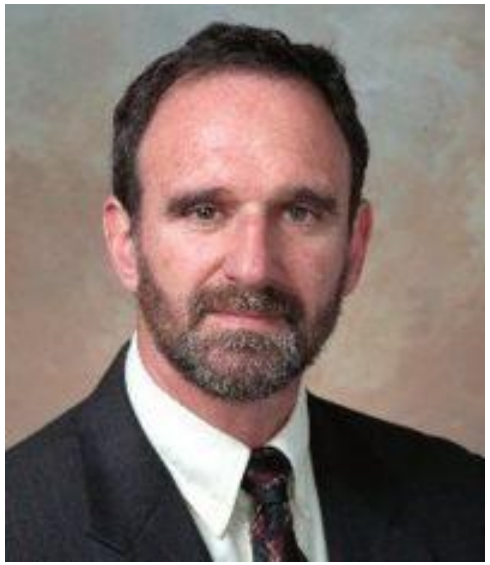
…Diffie-Hellman is the Solution

The Diffie-Hellman algorithm was invented in 1976 by Whitfield Diffie and Martin Hellman (US patent 4,200,770).

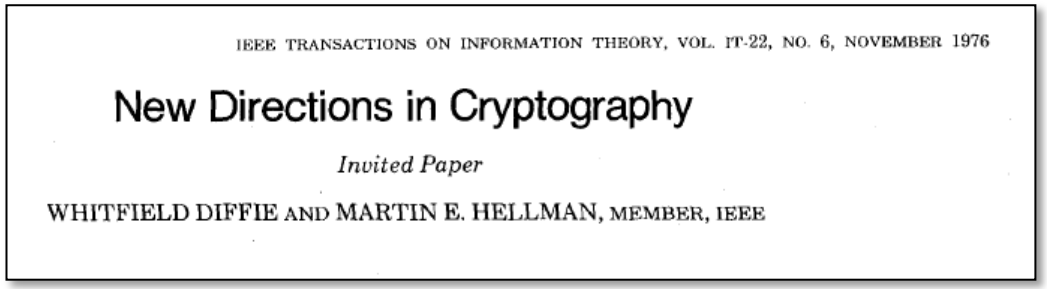
The Diffie-Hellman algorithm is not for encryption or decryption but it enables two parties who are involved in communication to generate a shared secret key for exchanging information confidentially.

They can do this even when an eavesdropper listens in on their entire conversation.

Whitfield 'Whit' Diffie



Martin Hellman



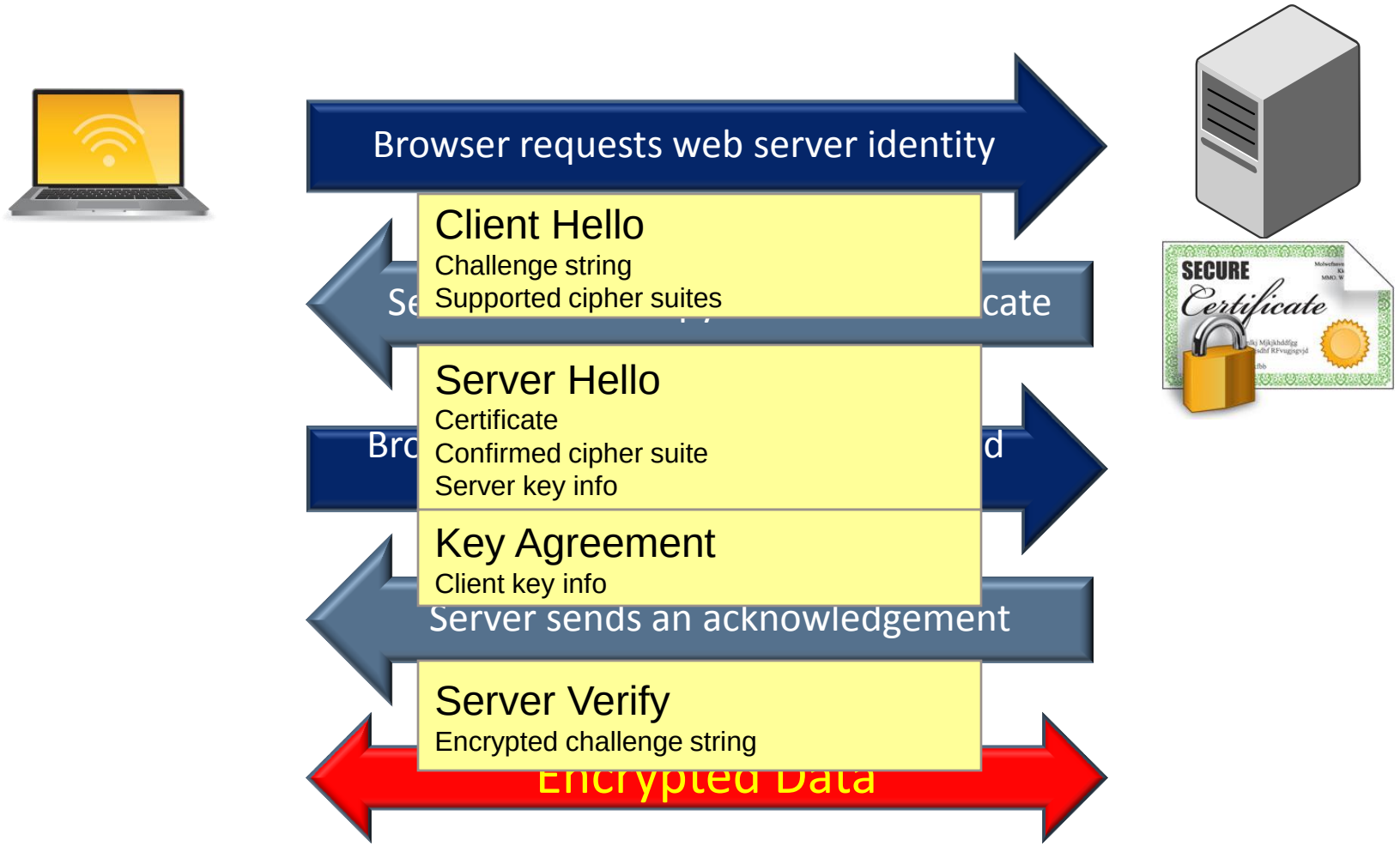
IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. IT-22, NO. 6, NOVEMBER 1976

New Directions in Cryptography

Invited Paper

WHITFIELD DIFFIE AND MARTIN E. HELLMAN, MEMBER, IEEE

SSL Handshake Process
Diffie-Hellman



Diffie-Hellman key agreement

- Using **Diffie-Hellman** key agreement, client and server agree a session key without ever sending the key across the network

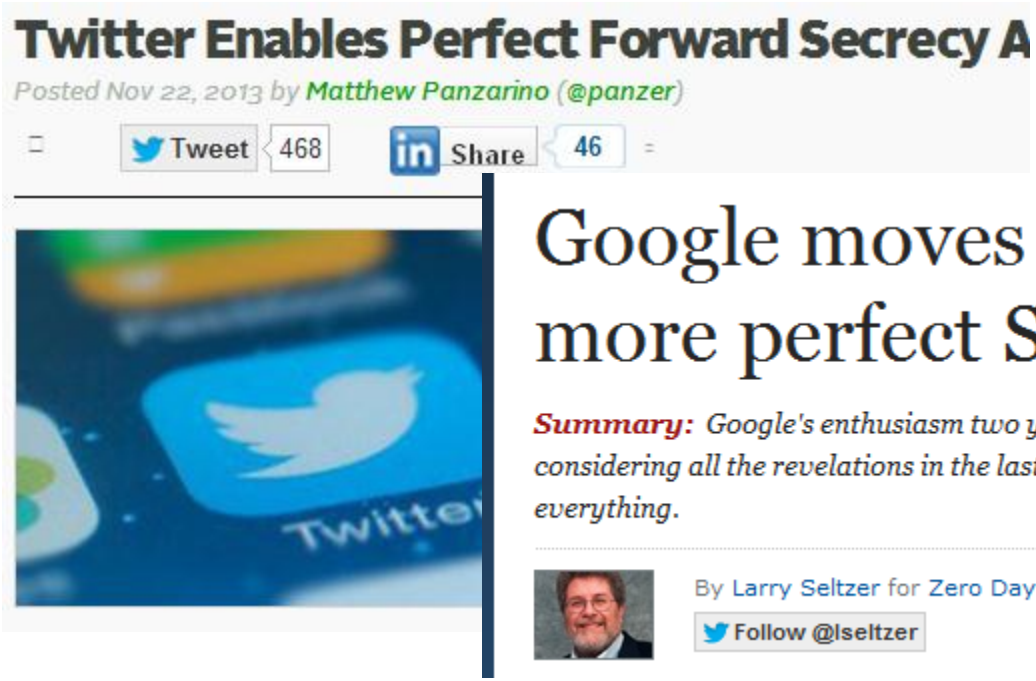
Diffie-Hellman can be used to enable Forward Secrecy

What is Perfect Forward Secrecy?

- **Forward Secrecy** uses Diffie-Hellman to generate a key for each session
- **Perfect Forward Secrecy** enhances Forward Secrecy by continuously changing the key material during each session
  - generating a new key for each message of a conversation

Where is Forward Secrecy Implemented?

- As of January 2014, 5% of TLS-enabled websites are configured to use cipher suites that provide forward secrecy to web browsers*

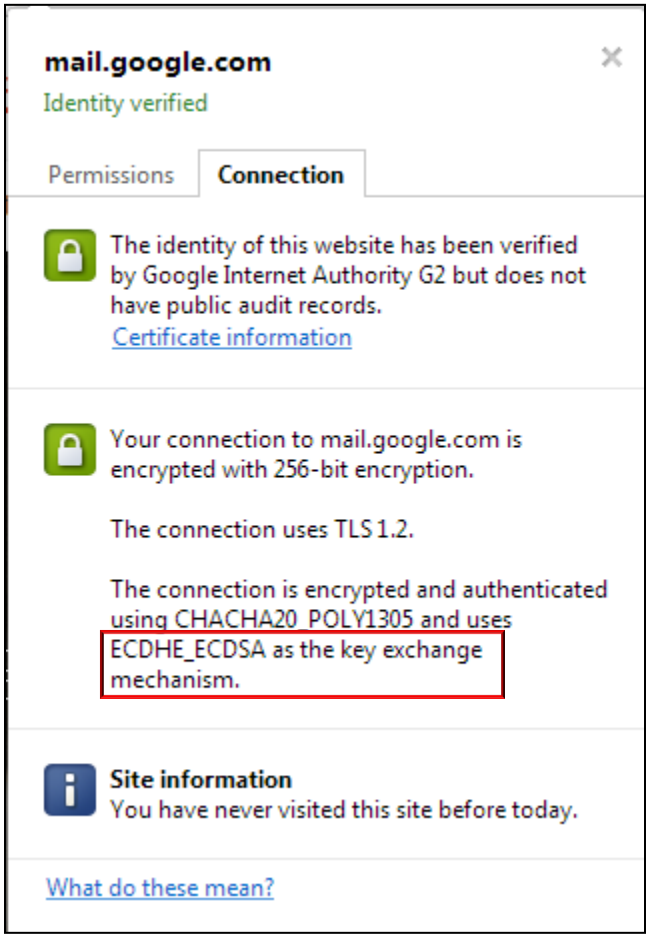


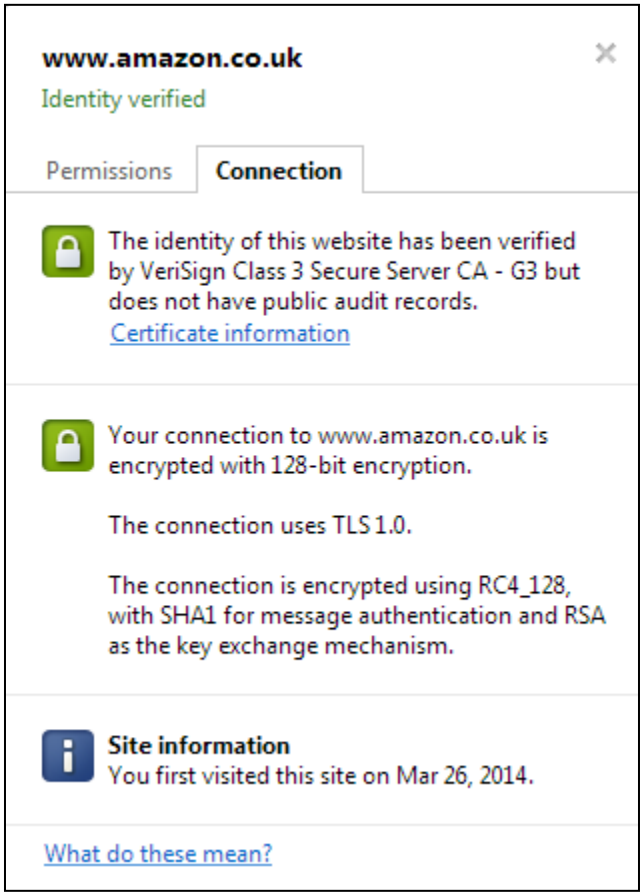
* SSL Pulse (January 03, 2014)

Are You Using Forward Secrecy?
Let's See…

Google™

Are You Using Forward Secrecy?



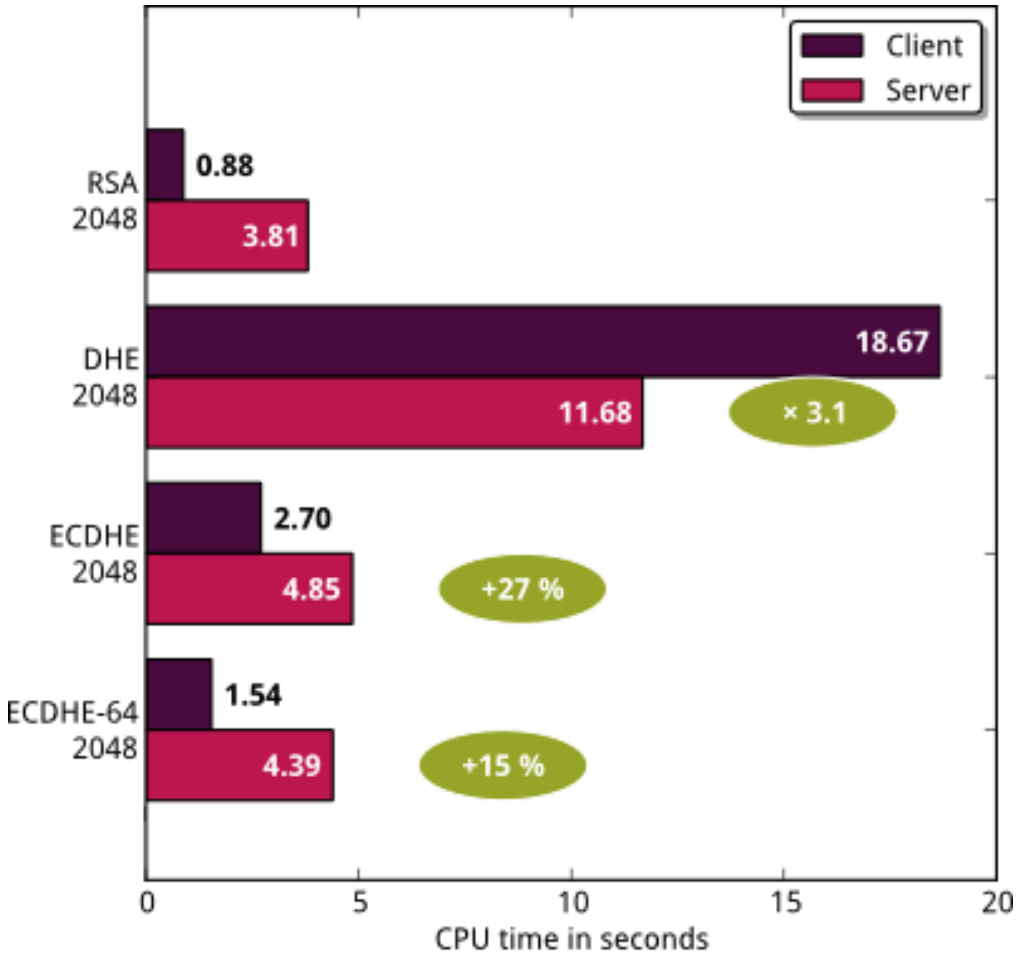


Implementing Forward Secrecy

http://www.theregister.co.uk/2013/06/26/ssl_forward_secrecy/

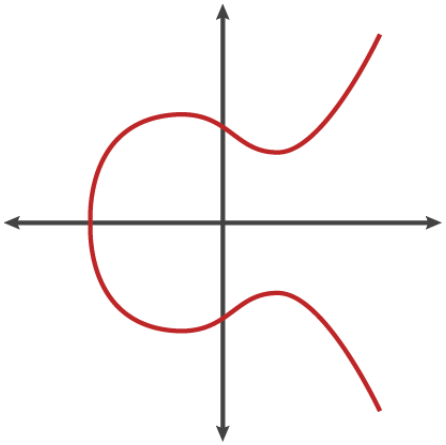
How do you implement Forward Secrecy?

- Diffie-Hellman Key Exchange (DHE) has a significant CPU overhead compared to RSA
- Elliptic Curve Diffie-Hellman (ECDHE) has minimal CPU overhead compared to RSA



Source: Bernat

Elliptic Curve Cryptography Overview



ECC

1 Stronger Encryption

- Shorter key than RSA
- 256-bit ECC = 3072-bit RSA
- 10k times harder to crack than RSA 2048
- Meets NIST recommendations

2 Efficient Performance

- Efficiency increases with higher server loads
- Utilizes less server CPU
- PC's: Faster page load time
- Ideal for mobile devices

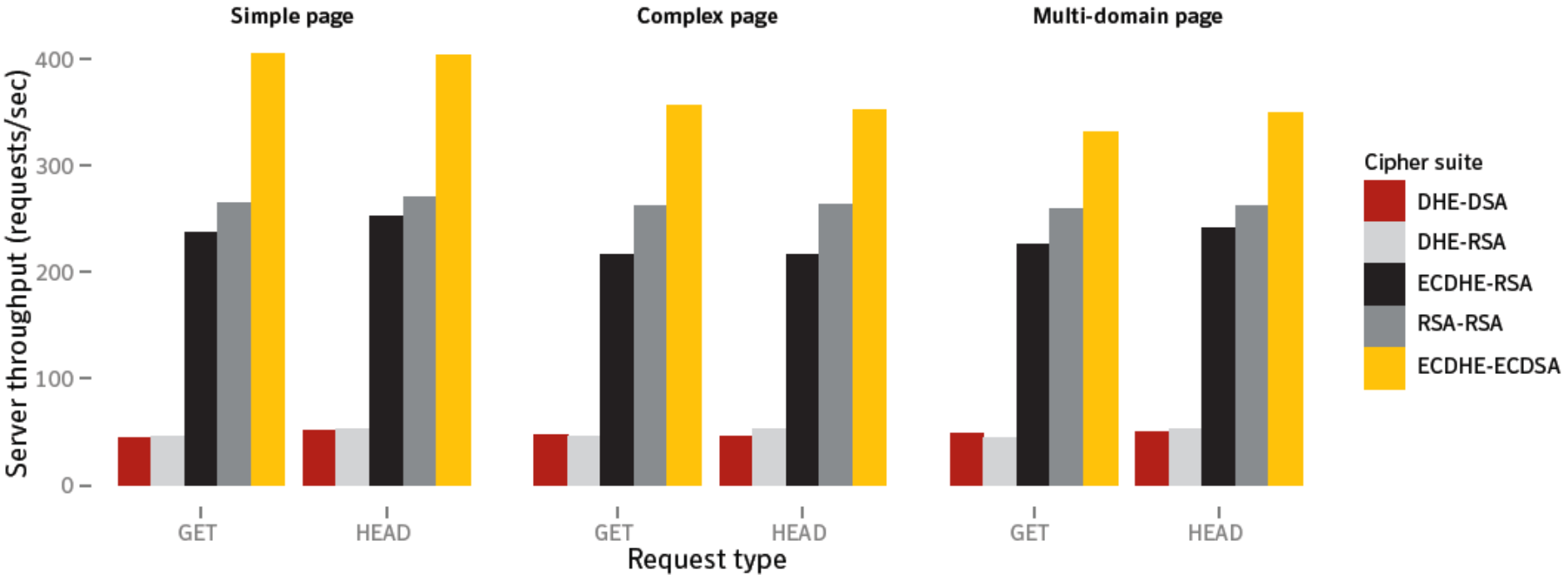
3 Highly Scalable

- Large SSL deployments w/out additional hardware
- Securing the enterprise:
  - Use fewer resources
  - Lower costs

4 Future of Crypto Tech

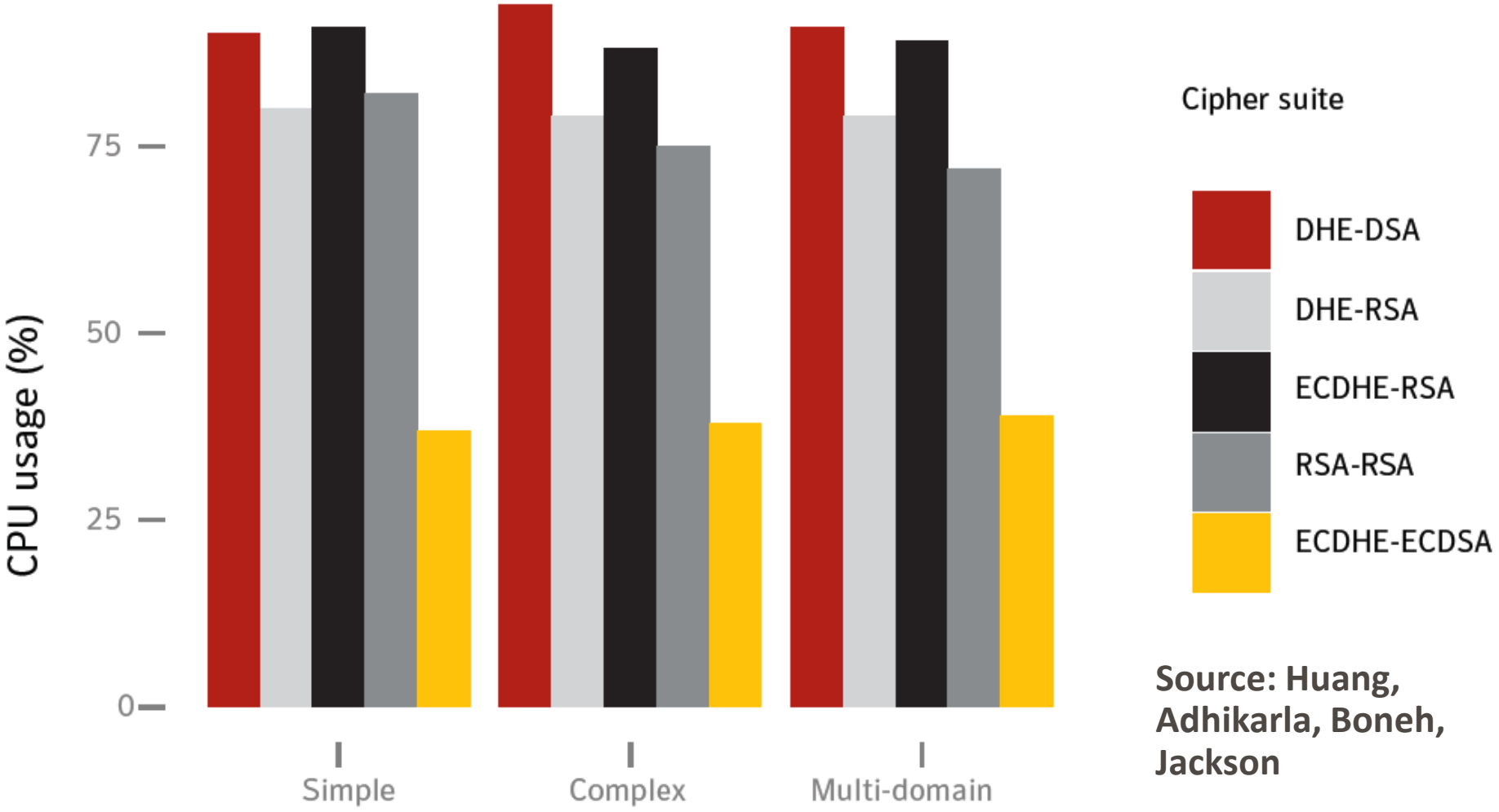
- Viable for many years
- Built for Internet of Things
- Supports billions of new devices coming online
- Ideal for Open Networks
- Truly "future proof" trust infrastructure in place

Implementing Forward Secrecy



Source: Huang, Adhikarla, Boneh, Jackson

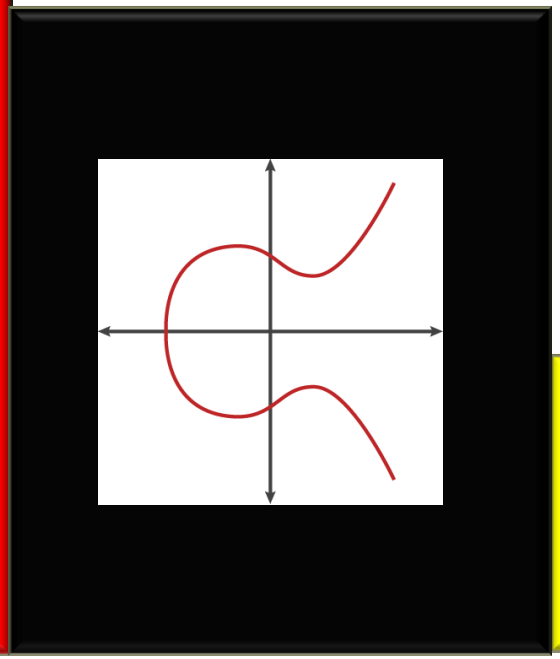
Implementing Forward Secrecy



Source: Huang, Adhikarla, Boneh, Jackson

Implementing Forward Secrecy

Use Diffie-Hellman key agreement for Forward Secrecy



Use Elliptic Curve Cryptography for Performance

Technical Overview

- What is the RSA Algorithm?
- What is Diffie-Hellman?
- What is Elliptic Curve Cryptography (ECC)?

RSA key exchange

- During the SSL handshake, the client receives the server's **certificate**
- This includes the server's **public key**
- Using **RSA**, the HTTPS **session key** is encrypted by the client using the server's **public key** and sent to the server across the network
- The server decrypts the session key using the corresponding **private key**

RSA is vulnerable if the private key is discovered

RSA Encryption & Decryption Example

Public key: n=33 e=3

Formula: $y=x^e \bmod n$

Encryption using public key				
Plaintext	we attack at dawn			
Binary plaintext	0101001001001011010 0			
Groups of 5 bits	01010	01001	00101	10100
Decimal plaintext (x)	10	9	5	20
x^3	1000	729	125	8000
Decimal ciphertext $y = x^3 \bmod 33$	10	3	26	14
Binary ciphertext	0101000011110100111 0			
Ciphertext	£orG^zplvnwk>?@qd			

RSA Encryption & Decryption Example

Private key: n=33 d=7

Formula: $x=y^d \bmod n$

Decryption using private key				
Ciphertext	£orG^zplvnwk>?@qd			
Binary ciphertext	01010000111101001110			
Decimal ciphertext y	10	3	26	14
y^7	10^7	2187	26^7	14^7
Decimal plaintext $x = y^7 \bmod 33$	10	9	5	20
Groups of 5 bits	01010	01001	00101	10100
Binary plaintext	0101001001010110100			
Plaintext	we attack at dawn			

RSA Encryption & Decryption Exercise

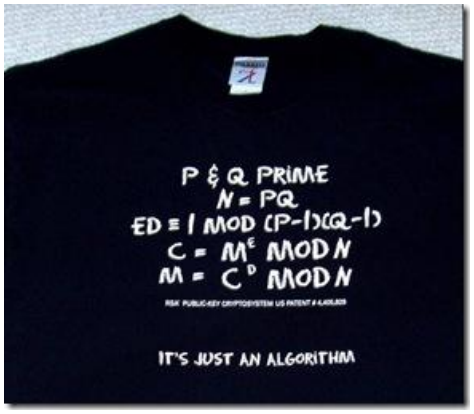
- Choose a number 1 < **x** < 33
- Encrypt x with RSA using the public key n=33, e=13
- Decrypt the encrypted number using the private key n=33, d=17
- Check if the decrypted number equals the original number

Encryption	**$y=x^e \bmod n$**
Decryption	**$x=y^d \bmod n$**

Modular Arithmetic (MOD Operation)

Modular arithmetic is related to that of the remainder in division. The operation of finding the remainder is sometimes referred to as the modulo operation, for example 14 (mod 12) = 2.

Why is RSA Secure?



p x **q** = **n**

Large prime numbers — Public Key

e

Private Key **d** = Function of **p** **q** **e**

Hard Problem:

To calculate *d* from *n* & *e*, you must find the *prime factors* of *n*

What are the prime factors of 6499?

What is 67 x 97?

How secure is RSA?

```
RSA-768 = 123018668453011775513049495838496272077285356959533479219732245215172640050726365751874520219978646938995647494277406384592519255732630345373154826850791702612214291346167042921431160222124047927473779408066535141959745985
          6902143413

RSA-768 = 33478071698956898786044169848212690817704794983713768568912431388982883793878002287614711652531743087737814467999489
        × 36746043666799590428244633799627952632279158164343087642676032283815739666511279233373417143396810270092798736308917
```

Asymmetric Key Length	Key Space	Equivalent Symmetric Key Length
768 bits	10^{232}	66 bits
1024 bits	10^{308}	80 bits
2048 bits	10^{616}	112 bits
3072 bits	10^{924}	128 bits

Diffie-Hellman key agreement

- Using **Diffie-Hellman** key agreement, client and server agree a session key without ever sending the key across the network

Diffie-Hellman can be used to enable Forward Secrecy

Diffie-Hellman Key Exchange Algorithm Example

Formula
$y = g^x \bmod n$

Public Parameters
Prime modulus: n=31
Primitive element: g=3

Random value a=8

$A = g^a \bmod n$
$= 3^8 \bmod 31$
$= 20$

Random value b=6

$B = g^b \bmod n$
$= 3^6 \bmod 31$
$= 16$

B=16

$S = B^a \bmod n$
$= 16^8 \bmod 31$
$= 4$

Shared Secret Key (S) is the same

A=20

$S = A^b \bmod n$
$= 20^6 \bmod 31$
$= 4$

$g^{ab} \bmod n = g^{ba} \bmod n$

Diffie-Hellman Key Exchange Exercise

- Choose a number $1 < \mathbf{x} < 20$
- Encrypt x with Diffie-Hellman using the public parameters n=31, g=3
- Exchange the encrypted number **y** with your neighbour
- Calculate the shared key (S) using your neighbour's encrypted number (y) and your number (x)
- Check with your neighbour if your values for S are the same

Encryption	$\mathbf{y=g^x \bmod n}$
Shared Key	$\mathbf{S=y^x \bmod n}$

Modular Arithmetic (MOD Operation)

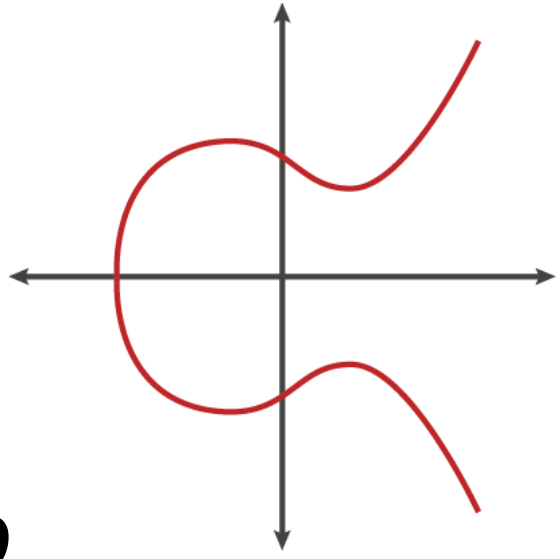
Modular arithmetic is related to that of the remainder in division. The operation of finding the remainder is sometimes referred to as the modulo operation, for example 14 (mod 12) = 2.

TLS 1.2 Key Exchange Mechanisms RFC 5246

Mechanism	Method	Server public key	Certificate signature
RSA	RSA	RSA	RSA
DH_RSA	Diffie-Hellman	Diffie-Hellman	RSA
DH_DSS	Diffie-Hellman	Diffie-Hellman	DSA
DHE_RSA	Ephemeral Diffie-Hellman	RSA	Any
DHE_DSS	Ephemeral Diffie-Hellman	DSA	Any
DH_anon	Diffie-Hellman	None	None

Forward Secrecy

Elliptic Curves

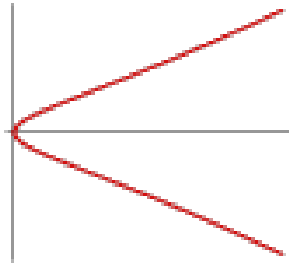


An *elliptic curve* is a plane curve defined by an equation of the form:

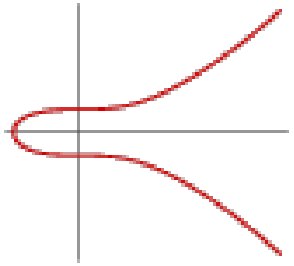
$$y^2 = x^3 + ax + b$$

Examples:

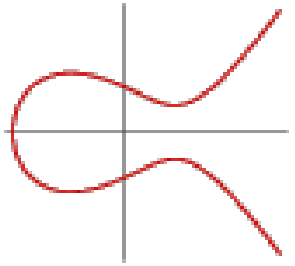
$y^2 = x^3 - 1$



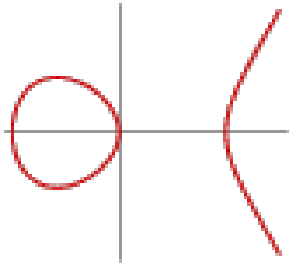
$y^2 = x^3 + 1$



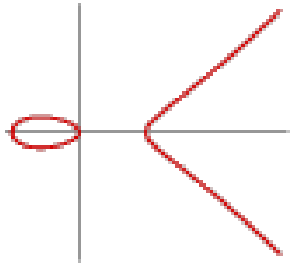
$y^2 = x^3 - 3x + 3$



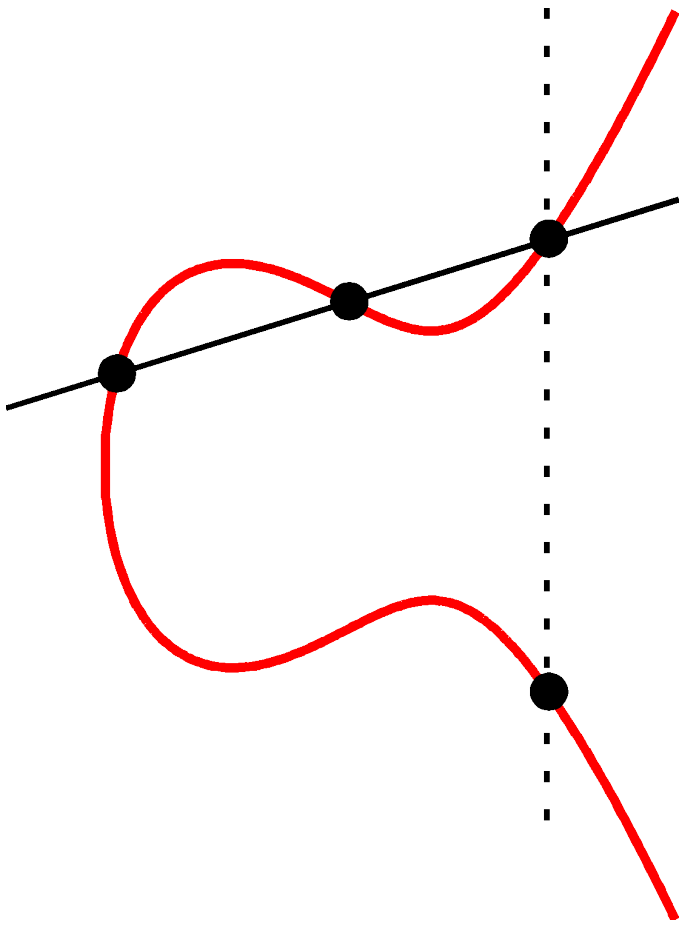
$y^2 = x^3 - 4x$



$y^2 = x^3 - x$

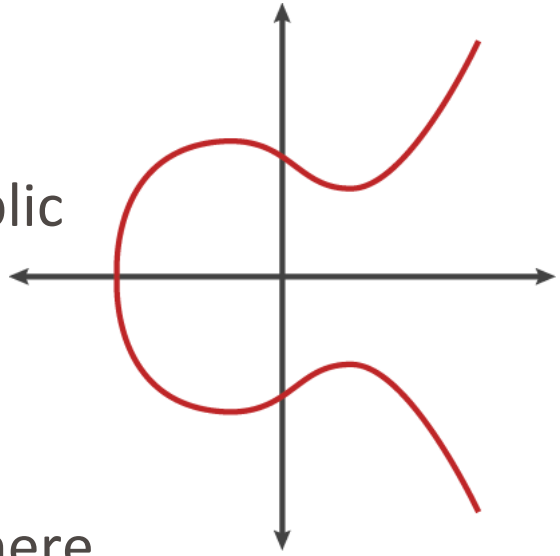


Elliptic Curve Cryptography (ECC)



- Adding 2 points on a curve produces another point on the curve
- Adding a point to itself produces another point on the curve
- If we add a point ***P*** to itself ***d*** times we end up with ***Q***, another point on the curve

Elliptic Curve Cryptography (ECC)



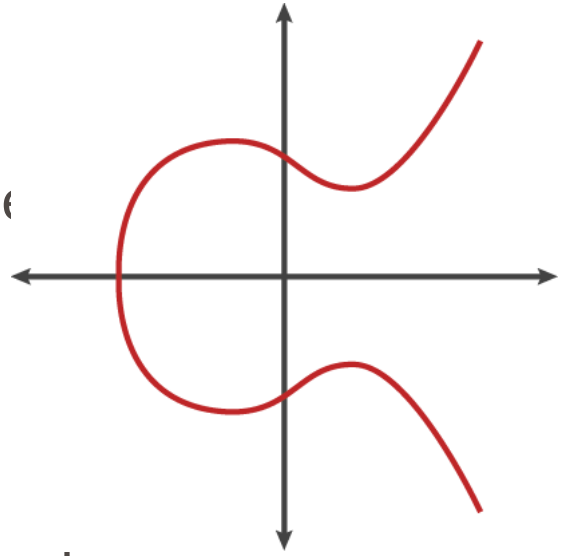
- Elliptic curves can be used to construct a public key cryptography system
- The private key ***d*** is randomly selected from [1,n-1], where n is integer
- Then the public key ***Q*** is computed by ***dP***, where P,Q are points on the elliptic curve
- Like the conventional cryptosystems, once the key pair (d, Q) is generated, a variety of cryptosystems such as signature, encryption/decryption and key management can be set up

Recommended Key Sizes

Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

http://www.nsa.gov/business/programs/elliptic_curve.shtml

Elliptic Curve Diffie-Hellman Example



- Alice and Bob make a key agreement over the following prime, curve, and point:

  $p=3851, E:y^2=x^3+324x+1287,$
  $P=(920,303)\in E(F_{3851})$

- Alice chooses the private key $\mathbf{d}_A=1194$, computes $Q_A=1194P=(2067,2178)\in E(F_{3851})$, and sends it to Bob
- Bob chooses the private key $\mathbf{d}_B=1759$, computes $Q_B=1759P=(3684,3125)\in E(F_{3851})$, and sends it to Alice
- Alice computes $d_AQ_B=1194(3684,3125)=(3347,1242)\in E(F_{3851})$
- Bob computes $d_BQ_A=1759(2067,2178)=(3347,1242)\in E(F_{3851})$

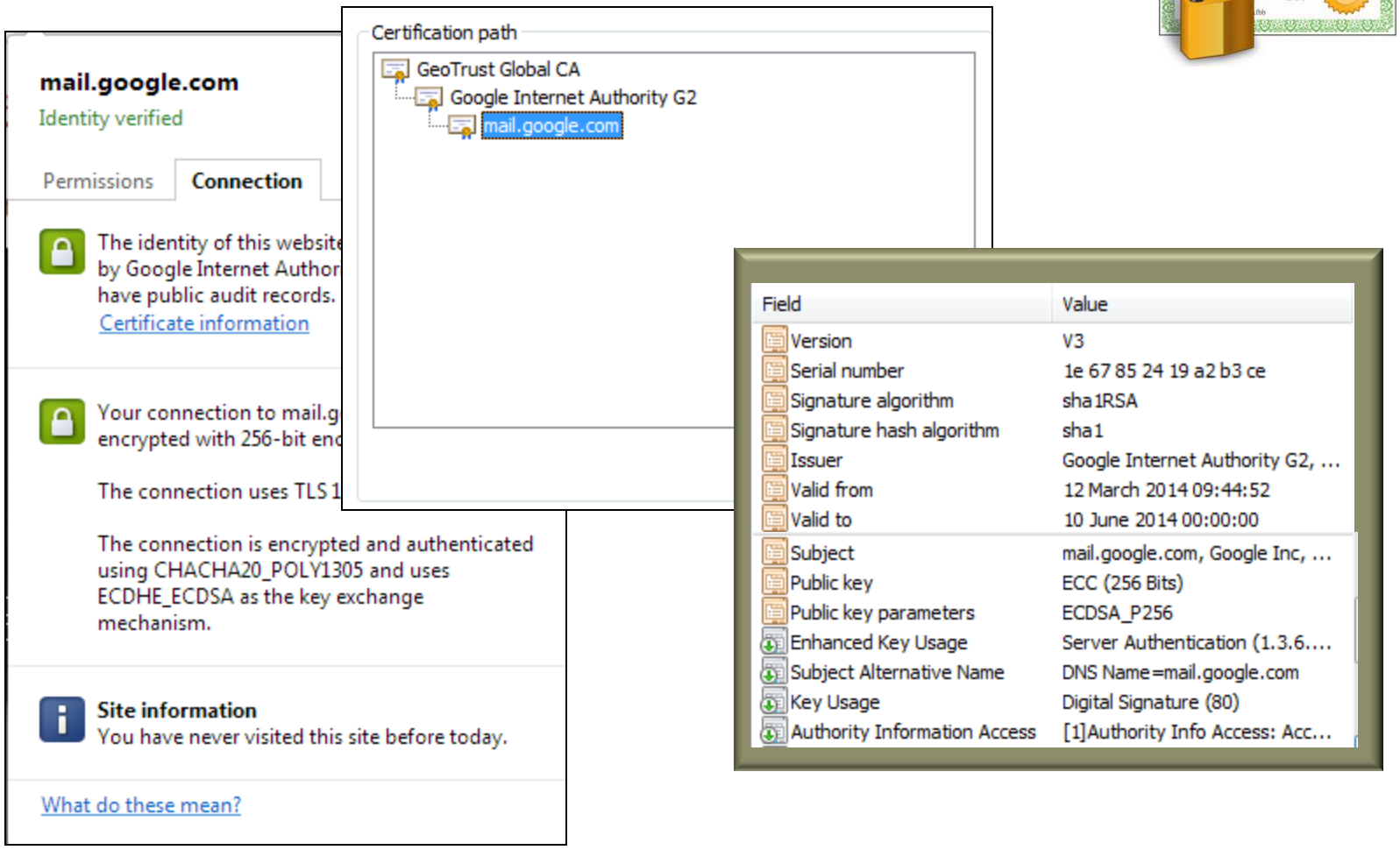
ECC Key Exchange Mechanisms
RFC 4492

Mechanism	Method	Server public key	Certificate signature
ECDH_RSA	Elliptic Curve Diffie-Hellman	Elliptic Curve Diffie-Hellman	RSA
ECDH_ECDSA	Elliptic Curve Diffie-Hellman	Elliptic Curve Diffie-Hellman	ECDSA
ECDHE_RSA	Ephemeral Elliptic Curve Diffie-Hellman	RSA	Any
ECDHE_ECDSA	Ephemeral Elliptic Curve Diffie-Hellman	ECDSA	Any
ECDH_anon	Ephemeral Elliptic Curve Diffie-Hellman	None	None

Forward Secrecy

ECC Certificate

ECC Certificate Example
Key Exchange Using ECDHE_ECDSA



Summary
Implementing Forward Secrecy

- Put ECDHE and DHE methods at top of server priority list
- Install ECDSA certificates

Mechanism	Method	Certificate	Performance
DHE_RSA	Ephemeral Diffie-Hellman	RSA	*
DHE_DSS	Ephemeral Diffie-Hellman	DSA	*
ECDHE_RSA	Ephemeral Elliptic Curve Diffie-Hellman	RSA	**
ECDHE_ECDSA	Ephemeral Elliptic Curve Diffie-Hellman	ECDSA	***

Use Symantec ECC Certificates to enable Forward Secrecy

An ECC certificate is included at no additional cost with all Symantec Premium SSL certificates

Symantec's ECC certificate roots have been in place for over five years: You can be confident that your ECC certificate will work throughout your ecosystem

Useful References

- Diffie–Hellman Key Agreement Method http://tools.ietf.org/html/rfc2631
- RSA: “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems" Communications of the ACM 21 (2): 120–126 http://people.csail.mit.edu/rivest/Rsapaper.pdf
- RFC4492 Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) http://tools.ietf.org/html/rfc4492
- FIPS 186-2 Digital Signature Standard (DSS) http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf

Useful Links

- Forward Secrecy at Twitter: https://blog.twitter.com/2013/forward-secrecy-at-twitter
- SSL Labs: Deploying Forward Secrecy: https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploying-forward-secrecy
- Symantec DSA / ECC FAQ: https://knowledge.verisign.com/support/ssl-certificates-support/index?page=content&id=SO21715&actp=search&viewlocale=en_US&searchid=1392807268836
- A (relatively easy to understand) primer on elliptic curve cryptography: http://arstechnica.com/security/2013/10/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/

A Final Note….

Public key cryptography was first discovered by American geeks. Right? Wrong.

- The Open Secret:
  - http://www.wired.com/wired/archive/7.04/crypto_pr.html
- The Alternative History of Public-Key Cryptography:
  - http://cryptome.org/ukpk-alt.htm
- Clifford Cocks & James Ellis:
  - http://en.wikipedia.org/wiki/Clifford_Cocks
  - http://en.wikipedia.org/wiki/James_H._Ellis

Thank you!

YOUR FEEDBACK IS VALUABLE TO US!

Please take a few minutes to fill out the short session survey available on the mobile app—the survey will be available shortly after the session ends. Watch for and complete the more extensive post-event survey that will arrive via email a few days after the conference.

To download the app, go to https://vision2014.quickmobile.com or search for Vision 2014 in the iTunes or Android stores.