CSc 466/566

Computer Security

7 : Cryptography — Public Key

Version: 2012/02/15 16:15:24

Department of Computer Science
University of Arizona

collberg@gmail.com

Christian Collberg

Outline

History of Public Key Cryptography

- RSA Conference 2011-Opening-Giants Among Us:
  http://www.youtube.com/watch?v=mvOsb9vNIWM&feature=related
- Rivest, Shamir, Adleman - The RSA Algorithm Explained:
  http://www.youtube.com/watch?v=b57zGAkNKIc
- Bruce Schneier - Who are Alice & Bob?:
  http://www.youtube.com/watch?v=BuUSi_QvFLY&feature=related
- Adventures of Alice & Bob - Alice Gets Lost:
  http://www.youtube.com/watch?v=nULAC_g22So http://www.youtube.com/watch?v=nJB7a79ahGM

Public-key Algorithms

Definition (Public-key Algorithms)

Public-key cryptographic algorithms use different keys for encryption and decryption.

- Bob's public key: P_B
- Bob's secret key: S_B

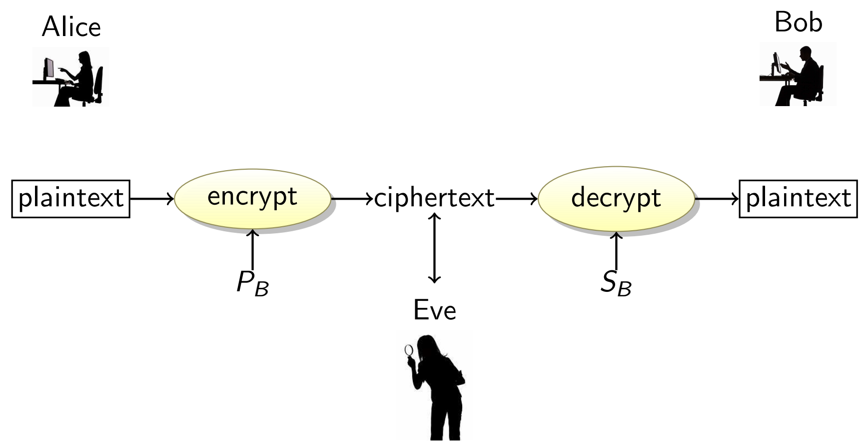
$$\begin{aligned} E_{P_B}(M) &= C \\ D_{S_B}(C) &= M \\ D_{S_B}(E_{P_B}(M)) &= M \end{aligned}$$

Public Key Protocol

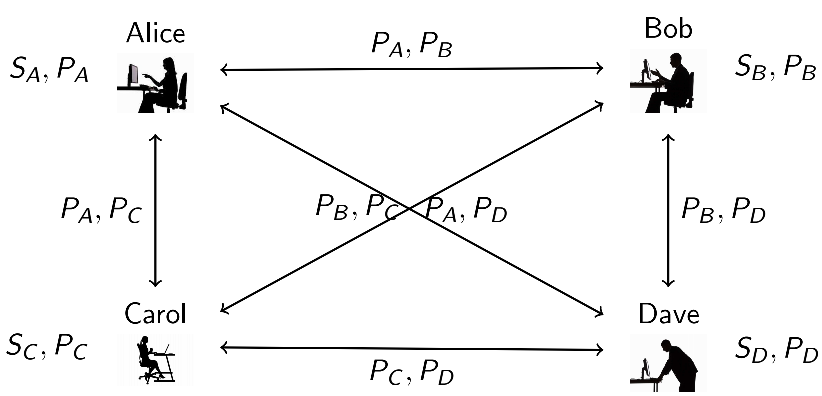
- Key-management is the main problem with symmetric algorithms – Bob and Alice have to somehow agree on a key to use.
- In public key cryptosystems there are two keys, a public one used for encryption and and private one for decryption.

1. Alice and Bob agree on a public key cryptosystem.
2. Bob sends Alice his public key, or Alice gets it from a public database.
3. Alice encrypts her plaintext using Bob's public key and sends it to Bob.
4. Bob decrypts the message using his private key.

Public Key Encryption Protocol. . .



Public Key Encryption: Key Distribution



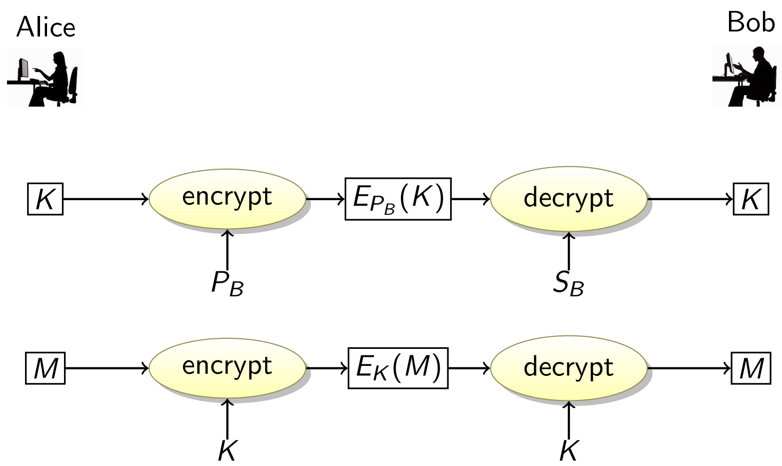
- Advantages: n key pairs to communicate between n parties.
- Disadvantages: Ciphers (RSA,. . .) are slow; keys are large

A Hybrid Protocol

- In practice, public key cryptosystems are not used to encrypt messages – they are simply too slow.
- Instead, public key cryptosystems are used to encrypt keys for symmetric cryptosystems. These are called session keys, and are discarded once the communication session is over.

1. Bob sends Alice his public key.
2. Alice generates a session key K, encrypts it with Bob's public key, and sends it to Bob.
3. Bob decrypts the message using his private key to get the session key K.
4. Both Alice and Bob communicate by encrypting their messages using K.

Hybrid Encryption Protocol. . .



Outline

RSA

- RSA is the best know public-key cryptosystem. Its security is based on the (believed) difficulty of factoring large numbers.
- Plaintexts and ciphertexts are large numbers (1000s of bits).
- Encryption and decryption is done using modular exponentiation.

RSA: Algorithm

- Bob (Key generation):
  1. Generate two large random primes p and q.
  2. Compute $n = pq$.
  3. Select a small odd integer e relatively prime with $\phi(n)$.
  4. Compute $\phi(n) = (p-1)(q-1)$.
  5. Compute $d = e^{-1} \bmod \phi(n)$.

  - $P_B = (e, n)$ is Bob's RSA public key.
  - $S_B = (d, n)$ is Bob' RSA private key.
- Alice (encrypt and send a message M to Bob):
  1. Get Bob's public key $P_B = (e, n)$.
  2. Compute $C = M^e \bmod n$.
- Bob (decrypt a message C received from Alice):
  1. Compute $M = C^d \bmod n$.

RSA: Algorithm Notes

- How should we choose e?
  - It doesn't matter for security; everybody could use the same e.
  - It matters for performance: 3, 17, or 65537 are good choices.
- n is referred to as the modulus, since it's the n of $\bmod n$.
- You can only encrypt messages $M < n$. Thus, to encrypt larger messages you need to break them into pieces, each $< n$.
- Throw away p, q, and $\phi(n)$ after the key generation stage.
- Encrypting and decrypting requires a single modular exponentiation.

RSA Example: Key Generations

1. Select two primes: $p = 47$ and $q = 71$.
2. Compute $n = pq = 3337$.
3. Compute $\phi(n) = (p-1)(q-1) = 3220$.
4. Select $e = 79$.
5. Compute

$$\begin{aligned} d &= e^{-1} \bmod \phi(n) \\ &= 79^{-1} \bmod 3220 \\ &= 1019 \end{aligned}$$

6. $P = (79, 3337)$ is the RSA public key.
7. $S = (1019, 3337)$ is the RSA private key.

RSA Example: Encryption

1. Encrypt $M = 6882326879666683$.
2. Break up M into 3-digit blocks:

$$m = \langle 688, 232, 687, 966, 668, 003 \rangle$$

   Note the padding at the end.
3. Encrypt each block:

$$\begin{aligned} c_1 &= m_1^e \bmod n \\ &= 688^{79} \bmod 3337 \\ &= 1570 \end{aligned}$$

   We get:

$$c = \langle 1570, 2756, 2091, 2276, 2423, 158 \rangle$$

RSA Example: Decryption

1. Decrypt each block:

$$\begin{aligned} m_1 &= c_1^d \bmod n \\ &= 1570^{1019} \bmod 3337 \\ &= 688 \end{aligned}$$

In-Class Exercise: Goodrich & Tamassia R-8.18

- Show the result of encrypting $M = 4$ using the public key $(e, n) = (3, 77)$ in the RSA cryptosystem.

In-Class Exercise: Goodrich & Tamassia R-8.20

- Alice is telling Bob that he should use a pair of the form
$$(3, n)$$
or
$$(16385, n)$$
as his RSA public key if he wants people to encrypt messages for him from their cell phones.
- As usual, $n = pq$, for two large primes, p and q.
- What is the justification for Alice's advice?

In-Class Exercise: Stallings pp. 270-271

1. Generate an RSA key-pair using $p = 17$, $q = 11$, $e = 7$.
2. Encrypt $M = 88$.
3. Decrypt the result from 2.

RSA Correctness

- We have
$$\begin{aligned} C &= M^e \bmod n \\ M &= C^d \bmod n. \end{aligned}$$
- To show correctness we have to show that decryption of the ciphertext actually gets the plaintext back, i.e that, for all $M < n$
$$\begin{aligned} C^d \bmod n &= (M^e)^d \bmod n \\ &= M^{ed} \bmod n \\ &= M \end{aligned}$$

RSA Correctness: Case 1

- From the key generation step we have

$$d = e^{-1} \bmod \phi(n)$$

from which we can conclude that

$$\begin{aligned} ed \bmod \phi(n) &= 1 \\ ed &= k\phi(n) + 1 \end{aligned}$$

- Case 1, M is relatively prime to n:

$$\begin{aligned} C^d \bmod n &= M^{ed} \bmod n \\ &= M^{k\phi(n)+1} \bmod n \\ &= M \cdot (M^{\phi(n)})^k \bmod n \\ &= M \cdot 1^k \bmod n \\ &= M \bmod n \\ &= M \end{aligned}$$

RSA Correctness: Case 1...

- $M^{\phi(n)} \bmod n = 1$ follows from Euler's theorem.

Theorem (Euler)

Let x be any positive integer that's relatively prime to the integer $n > 0$, then

$$x^{\phi(n)} \bmod n = 1$$

RSA Correctness: Case 2

- Assume that M is *not* relatively prime to n, i.e. M has some factor in common with n, since $M < n$.
- There are two cases:
  1. M is relatively prime with q and $M = ip$, or
  2. M is relatively prime with p and $M = iq$.
- We consider only the first case, the second is similar.

RSA Correctness: Case 2...

- We have that

$$\phi(n) = \phi(pq) = \phi(p)\phi(q)$$

- By Euler's theorem we have that

$$\begin{aligned} M^{k\phi(n)} \bmod q &= M^{k\phi(p)\phi(q)} \bmod q \\ &= (M^{k\phi(p)})^{\phi(q)} \bmod q \\ &= 1 \end{aligned}$$

- Thus, for some integer h

$$M^{k\phi(n)} = 1 + hq$$

- Multiply both sides by M

$$\begin{aligned} M \cdot M^{k\phi(n)} &= M(1 + hq) \\ M^{k\phi(n)+1} &= M + Mhq \end{aligned}$$

RSA Correctness: Case 2...

- We can now prove Case 2, for $M = ip$:

$$\begin{aligned}
C^d \bmod n &= M^{ed} \bmod n \\
&= M^{k\phi(n)+1} \bmod n \\
&= (M + Mhq) \bmod n \\
&= (M + (ip)hq) \bmod n \\
&= (M + (ih)pq) \bmod n \\
&= (M + (ih)n) \bmod n \\
&= (M \bmod n) + ((ih)n \bmod n) \\
&= M \bmod n \\
&= M
\end{aligned}$$

RSA Security

- Summary:
  1. Compute $n = pq$, p and q prime.
  2. Select a small odd integer e relatively prime with $\phi(n)$.
  3. Compute $\phi(n) = (p-1)(q-1)$.
  4. Compute $d = e^{-1} \bmod \phi(n)$.
  5. $P_B = (e, n)$ is Bob's RSA public key.
  6. $S_B = (d, n)$ is Bob' RSA private key.
- Since Alice knows Bob's P_B, she knows e and n.
- If she can compute d from e and n, she has Bob's private key.
- If she knew $\phi(n) = (p-1)(q-1)$ she could compute $d = e^{-1} \bmod \phi(n)$ using Euclid's algorithm.
- If she could factor n, she'd get p and q!

Security of Cryptosystems by Failed Cryptanalysis

1. Propose a cryptographic scheme.
2. If an attack is found, patch the scheme. GOTO 2.
3. If enough time has passed ⇒ The scheme is secure!

- How long is enough?
  1. It took 5 years to break the Merkle-Hellman cryptosystem.
  2. It took 10 years to break the Chor-Rivest cryptosystem.

RSA Security...

- If we can factor n, we can find p and q and the scheme is broken.
- As far as we know, factoring is hard.
- We need n to be large enough, 2,048 bits.

RSA Factoring Challenge

http://www.rsa.com/rsalabs/node.asp?id=2093

```
Name:        RSA-576
Digits:      174
188198812920607963838697239461650439807163563379417382700763356422
988859715234665485319060606504743045317388011303396716199692321205
734031879550656996221305168759307650257059
```

- On December 3, 2003, a team of researchers in Germany and several other countries reported a successful factorization of the challenge number RSA-576.
- The factors are

```
3980750864240649373971255005503864911990643623
42526708406385189575946388957261768583317

47277214610743530253622307197304822463291469530
209711645985217113052071125636359039752 7
```


RSA Factoring Challenge. . .

```
Name:        RSA-640
Digits:      193
310741824049004372135075003588856793003734602284272754572016194882
320644051808150455634682967172328678243791627283803341547107310850
191954852900733772482278352574238645401469173660247765234660 9
```

- The factoring research team of F. Bahr, M. Boehm, J. Franke, T. Kleinjung continued its productivity with a successful factorization of the challenge number RSA-640, reported on November 2, 2005.
- The factors are:

```
16347336458092538484431338838650908598417836700330
92312181110852389333100104508151212118167511579

1900871281664822113126851573935413975471896789968
515493666638539088027103802104498957191261465571
```

- The effort took approximately 30 2.2GHz-Opteron-CPU years according to the submitters, over five months of calendar time.

RSA Factoring Challenge. . .

```
Name:        RSA-704
Digits:      212
740375634795617128280467960974295731425931888892312890849362326389 7
276503402826627689199641962511784399589433050212758537011896809828 6
733173273108930900552505116877063299072396380786710086096962537934650563796359

Name:        RSA-768
Digits:      232
123018668453011775513049495838496272077285356959533479219732245215172
640050726365751874520219978646938995647494277406384592519255732630345
373154826850791702612214291346167042921431160222124047927473779408066 5
35141959745985690214341 3

Name:        RSA-896
Digits:      270
412023436986659543855531365332575948179811699844327982845455626433876 4
455652484261980988704231618418792614202471888694925609317763750334211 3
098239748515094490910691026986103186270411488086697056490290365365886 7
433731720813104105190864254793282601391257624033946373269391

Name:        RSA-1024
Digits:      309
135066410865995223349603216278805969938881475605667027524485143851526 5
106048595338339402871505719094417982072821644715513736804197039641917 4
304649658927425623934102086438320211037295872576235850964311056407350 1
508187510676594629205563685529475213500852879416377328533906109750544 3
3499981115005697723689092756 3
```


RSA Factoring Challenge. . .

```
Name:        RSA-1536
Digits:      463
184769970321174147430683562020016440301854933866341017147178577491065 1
696711161249859337684305435744585616061544571794052229717732524660960 6
469460712496237204420222697567566873784275623895087646784409332851574 9
657884341508847552829818672645133986336493190808467199043187438128336 3
502795470282653297802934916155811881049844908319545009848393775227257 0
525785919449938700736957556884369338127796130892303925696952532616208 2
367649031603655137144791393234716956698806 9

Name:        RSA-2048
Digits:      617
251959084756578934940271832400483985714292821262040320277771378360436 6
202070759555626401852588078440691829064124951508218929855914917618450 2
808489120072844992687392807287776735971418347270261896375014971824691 1
650776133798590957000973304597488084284017974291006424586918171951187 4
612151517265463228221686998754918242243363725908514186546204357679842 3
387184774447920739934236584823824281198163815010674810451660377306056 2
016196762561338441436038339044149526344321901146575444541784240209246 1
651572335077870774981712577246796292638635637328991215483143816789988 5
040445364023527381951378636564391212010397122822120720357
```

RSA Security: How to use RSA

- Two plaintexts M_1 and M_2 are encrypted into ciphertexts C_1 and C_2.
- But, RSA is deterministic!
- If $C_1 = C_2$ then we know that $M_1 = M_2$!
- Also, side-channel attacks are possible against RSA, for example by measuring the time taken to encrypt.

Outline

Software – GPG

- gpg is a public domain implementation of pgp.
- Supported algorithms:
  - Pubkey: RSA, RSA-E, RSA-S, ELG-E, DSA
  - Cipher: 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH, CAMELLIA128, CAMELLIA192, CAMELLIA256
  - Hash: MD5, SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
  - Compression: Uncompressed, ZIP, ZLIB, BZIP2
- http://www.gnupg.org.

Key generation: Bob

```
> gpg --gen-key

Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
What keysize do you want? (2048)
Key is valid for? (0)
Key does not expire at all
Real name: Bobby
Email address: bobby@gmail.com
Comment: recipient
You need a Passphrase to protect your secret key.
Enter passphrase: Bob rocks
Repeat passphrase: Bob rocks
```

Key generation: Alice

```
> gpg --gen-key

Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
What keysize do you want? (2048)
Key is valid for? (0)
Key does not expire at all
Real name: Alice
Email address: alice@gmail.com
Comment: sender
You need a Passphrase to protect your secret key.
Enter passphrase: Alice is cute
Repeat passphrase: Alice is cute
```


Exporting the Key

```
> gpg --armor --export Bobby
-----BEGIN GPG PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.11 (Darwin)

mQENBE83U28BCADTVOkHpNjWzk7yEzMhiNJcmOtmUYfn4hzgYTDsP2otIOUhfJ4q
EZCuPoxECIZ479k3YpBvZM2JC48Ht9j1kVnDPLCrongyRdSko0AwG7OYAyHWa7/U
SeGwjZ+0MUuM3SwqHdo1/0XS3P8LABTQNXtrQf9kF8UNLIaHr1IvBcae1K44MPL6
................................................................
EBHmAM7iiWgWI6/6qEmN46ZQEmoR86vWhQL3LQ6p/FUaBA==
=FZ78
-----END GPG PUBLIC KEY BLOCK-----
```


Encryption

- We can encrypt a message using Bobby's key:

```
> cat message
Attack at dawn
> gpg --recipient bobby --armor --encrypt message
> cat message.asc
-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.4.11 (Darwin)

hQEMA97v9lbZUpHvAQf/a9QklXMiMzBWy5yyZBtNrg7FcrIqx+gXVVUXNN86tZtE
RF42elwU6QwamDzfcOHqp+3zeor4Y5xN+/pL91xti6uwFOhgGrCGJq//AfUKgQyk
MH2e4gR8Y1BuPm9b1c7uzXxRMMOUBBt75KquYGOBLybsP29ttD9iL/ZJl1zSPjSj
El7OOGp7PqEBotStVOtuknYW/fX0zXndU8XNllKnsnZn21Xm0rMQcFMu8Do/tF5I
lRfTEcL4S9tV4vshgXhNSpTg9sZs1UZynvU2cJqyYkCtgT7TdtrK3fTa8UN+CYQv
U2QRnaNtFhYwBMonFqhefNzDqeZb+P0RqOuoDllYuNJRAViJ3CLjT7kwgBgRtNfY
RkGArQQmgrknW2jq/Y2GZTE8CC7pNXY8U3KYMl9hRA6U5fMp08ndFp8vowBbB2sw
zjxjSY7ZeIR2uwxdLYydtW4m
=B+JA
-----END PGP MESSAGE-----
```


Decryption

- Bobby can now decrypt the message using his private key:

```
> gpg --decrypt message.asc

You need a passphrase to unlock the secret key for
user: "Bobby (recipient) <bobby@gmail.com>"
2048-bit RSA key, ID D95291EF, created 2012-02-12
(main key ID 9974031B)

Enter passphrase: Bob rocks

gpg: encrypted with 2048-bit RSA key, ID D95291EF, created 2012-02-12
      "Bobby (recipient) <bobby@gmail.com>"
Attack at dawn
```

The keyring

```
> gpg --list-keys
/Users/collberg/.gnupg/pubring.gpg
----------------------------------
pub   2048R/9974031B 2012-02-12
uid                  Bobby (recipient) <bobby@gmail.com>
sub   2048R/D95291EF 2012-02-12

pub   2048R/4EC8A0CB 2012-02-12
uid                  Alice (sender) <alice@gmail.com>
sub   2048R/B901E082 2012-02-12
```


The keyring. . .

```
> gpg --list-secret-keys
/Users/collberg/.gnupg/secring.gpg
----------------------------------
sec   2048R/9974031B 2012-02-12
uid                  Bobby (recipient) <bobby@gmail.com>
ssb   2048R/D95291EF 2012-02-12

sec   2048R/4EC8A0CB 2012-02-12
uid                  Alice (sender) <alice@gmail.com>
ssb   2048R/B901E082 2012-02-12
```


Sign and Encrypt

- Bob can sign his message before sending it to Alice:

```
> gpg -se --recipient alice --armor message

You need a passphrase to unlock the secret key for
user: "Bobby (recipient) <bobby@gmail.com>"
2048-bit RSA key, ID 9974031B, created 2012-02-12

Enter passphrase: Bob rocks

> cat message.asc
-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.4.11 (Darwin)

hQEMA7osp1S5AeCCAQgAsSqSs+Urf0f3KHTtP7cqTwugpcJ9oUAGkw/KQODHIE0v
.................................................................
8XEAaCwZ8aZK1lXhqBSd/9hCm9Mup2NECihO8crVyff7NTWFyaTBeGAm10q3y46o
QpIgPbcdYZqIt8e/8wPU6xlMZUStzxBKLB+Rj/Zg35ZVioYL
=oiv8
-----END PGP MESSAGE-----
```


Check Signature and Decrypt

- Alice can now decrypt the message and check the signature:

```
> gpg --decrypt message.asc

You need a passphrase to unlock the secret key for
user: "Alice (sender) <alice@gmail.com>"
2048-bit RSA key, ID B901E082,
created 2012-02-12 (main key ID 4EC8A0CB)

Enter passphrase: Alice is cute

gpg: encrypted with 2048-bit RSA key, ID B901E082, created 2012-02-12
      "Alice (sender) <alice@gmail.com>"
Attack at dawn
gpg: Signature made Sat Feb 11 23:10:59 2012 MST
using RSA key ID 9974031B
gpg: Good signature from "Bobby (recipient) <bobby@gmail.com>"
```

Symmetric Encryption Only

```
> gpg --cipher-algo=AES --armor --symmetric message
Enter passphrase: sultana
Repeat passphrase: sultana
> cat message.asc
-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.4.11 (Darwin)

jA0EBwMCgZ3PBfSZxJlg0ksBBooTMLEVQ2q9HkTR5y9FIoX9nbsyohr0XeQLFlcf
wtWcg+dZvlMS6D70E3wZCeW2LX50kYcU17MUc8wnJLDAzAdRqPAgDma+sP4=
=UtI4
-----END PGP MESSAGE-----

> gpg message.asc
gpg: AES encrypted data
Enter passphrase: sultana
gpg: encrypted with 1 passphrase

> cat message
Attack at dawn
```


Deleting Keys

```
> gpg --delete-secret-keys bobby
sec  2048R/9974031B 2012-02-12 Bobby (recipient) <bobby@gmail.com>

Delete this key from the keyring? (y/N) y
This is a secret key! - really delete? (y/N) y

> gpg --delete-keys bobby
pub  2048R/9974031B 2012-02-12 Bobby (recipient) <bobby@gmail.com>

Delete this key from the keyring? (y/N) y
```


Generating Primes

- Generate a prime number of the given number of bits:

```
> gpg --gen-prime 1 16
C4B7
> gpg --gen-prime 1 1024
D34D4347ED013242EE06811BC561C6587D75ADE33D1BEC954D648E22
9D88B5E0AF1394459FB48B135B99C8BA8C50E5331C6226CBF6D70031
4A8CC84C7B363BE7DD7BBBB29E545D199339263F5FB2E9F1B84BA9D5
05B5B79858FC6149CF09E6C56D9730C3BD1E62B378C8DFAF4233B8DC
BA999A21EC9C4BF8C60AACDCBC607AC5
```


Generating Random Numbers

- Generate 100 (base64 encoded) random bytes:

```
> gpg --armour --gen-random 0 100
e0zAVl6jbe/Dma9VF20lMgZxE1RA4S8TwNwu6KP8+o1kjdtBm2
AjKFSVsj/d3zG/9KqmNj7j6symEUZ3e0fWZaWqLBxzJuSur5sK
C8omfPus2QtYJJN0gVbpJ7X9L4t1iNJtnw==
```

Print Message Digests

```
> gpg --print-mds message
MD5    = 36 D1 A5 12 17 CD 34 FC 04 F5 6C C4 91 39 C7 59
SHA1   = 6DA4 473A 00CE 7AB6 7B6F 884D 1E75 6633 C21A 56DB
RMD160 = D1DE 4194 C0CD 3AED 30F3 38CD 68F3 800F CCF0 3B87
SHA224 = B4E94780 1AA1A9C3 418F72D8 651BA995 83284003
         EBEE183A 589702EE
SHA256 = B83EF405 07696578 9D4BBDA7 D7932700 5F2AE6CB
         A2696FDE 69694D12 AFE70E4A
SHA384 = 7AC39A0C 945844F1 1316BB46 C9FC7EEA E892A178
         2D20E4CA E7BE686C 1A091C8C F1BBDFD1 3D42BEA2
         88AF5A4F E3705474
SHA512 = 9CA1EB88 F064CB0D 536254B2 5755919F 45564276
         96CA27A0 389E4817 53F81DC2 3222488D 7D11F3DD
         C066B9E8 027F3870 395A2561 157DDC38 BD679D37
         C2E361CC
```


Goal: Read a message encrypted with gpg

1. Decrypt the message itself (OR)
2. Determine symmetric key used to encrypt the message by other means (OR)
3. Get recipient to help decrypt message (OR)
4. Obtain private key of recipient.

http://www.schneier.com/paper-attacktrees-fig7.html

Goal: Read a message encrypted with gpg. . .

Decrypt the message itself:

1. Break asymmetric encryption (OR)
   1. Brute force break asymmetric encryption (OR)
   2. Mathematically break asymmetric encryption (OR)
      1. Break RSA (OR)
      2. Factor RSA modulus/calculate Elgamal discrete log
   3. Cryptanalyze asymmetric encryption (OR)
      1. General cryptanalysis of RSA/Elgamal (OR)
      2. Exploit weakness in RSA/Elgamal (OR)
      3. Timing attack on RSA/Elgamal
2. Break symmetric-key encryption
   1. Brute force break symmetric-key encryption
   2. Cryptanalysis of symmetric-key encryption

Goal: Read a message encrypted with gpg. . .

Determine symmetric key by other means:

1. Fool sender into encrypting message using public key whose private key is known (OR)
   1. Convince sender that fake key (with known private key) is the key of the intended recipient
   2. Convince sender to encrypt with more than one key—the real key of the recipient and a key whose private key is known.
   3. Have the message encrypted with a different public key in the background, unbeknownst to the sender.
2. Have the recipient sign the encrypted publc key (OR)
3. Monitor the sender's computer memory (OR)
4. Monitor the receiver's computer memory (OR)
5. Determine key from pseudo-random number generator (OR)
   1. Determine state of randseed during encryption (OR)
   2. Implant virus that alters the state of randseed. (OR)
   3. Implant software that affects the choice of symmetric key.
6. Implant virus that that exposes public key.

Goal: Read a message encrypted with gpg...

Get recipient to help decrypt message:

Goal: Read a message encrypted with gpg...

Obtain private key of recipient:

Goal: Read a message encrypted with PGP

What immediately becomes apparent from the attack tree is that breaking the RSA or IDEA encryption algorithms are not the most profitable attacks against PGP. There are many ways to read someone's PGP-encrypted messages without breaking the cryptography. You can capture their screen when they decrypt and read the messages (using a Trojan horse like Back Orifice, a TEMPEST receiver, or a secret camera), grab their private key after they enter a passphrase (Back Orifice again, or a dedicated computer virus), recover their passphrase (a keyboard sniffer, TEMPEST receiver, or Back Orifice), or simply try to brute force their passphrase (I can assure you that it will have much less entropy than the 128-bit IDEA keys that it generates).

Goal: Read a message encrypted with PGP...

In the scheme of things, the choice of algorithm and the key length is probably the least important thing that affects PGP's overall security. PGP not only has to be secure, but it has to be used in an environment that leverages that security without creating any new insecurities.

http://www.schneier.com/paper-attacktrees-fig7.html

Outline

Elgamal

- The Elgamal cryptosystem relies on the inherent difficulty of calculating discrete logarithms.
- It is a probabilistic scheme:
  - a particular plaintext can be encrypted into multiple different ciphertexts;
  - $\Rightarrow$ ciphertexts become twice the length of the plaintext.

Elgamal: Algorithm

- Bob (Key generation):
  1. Pick a prime p.
  2. Find a generator g for Z_p.
  3. Pick a random number x between 1 and $p-2$.
  4. Compute $y = g^x \bmod p$.
  - $P_B = (p, g, y)$ is Bob's RSA public key.
  - $S_B = x$ is Bob' RSA private key.
- Alice (encrypt and send a message M to Bob):
  1. Get Bob's public key $P_B = (p, g, y)$.
  2. Pick a random number k between 1 and $p-2$.
  3. Compute the ciphertext $C = (a, b)$:

$$\begin{aligned} a &= g^k \bmod p \\ b &= My^k \bmod p \end{aligned}$$

- Bob (decrypt a message $C = (a, b)$ received from Alice):
  1. Compute $M = b(a^x)^{-1} \bmod p$.

Elgamal: Algorithm Notes

- Alice must choose a different random number k for every message, or she'll leak information.
- Bob doesn't need to know the random value k to decrypt.
- Each message has $p-1$ possible different encryptions.
- The division in the decryption can be avoided by use of Lagrange's theorem:

$$\begin{aligned} M &= b \cdot (a^x)^{-1} \bmod p \\ &= b \cdot a^{p-1-x} \bmod p \end{aligned}$$

Elgamal: Finding the generator

- Computing the generator is, in general, hard.
- We can make it easier by choosing a prime number with the property that we can factor $p-1$.
- Then we can test that, for each prime factor p_i of $p-1$:

$$g^{(p-1)/p_i} \bmod p \neq 1$$

  If g is not a generator, then one of these powers will $\neq 1$.

Elgamal Example: Key generation

1. Pick a prime $p = 13$.
2. Find a generator $g = 2$ for Z_{13} (see next slide).
3. Pick a random number $x = 7$.
4. Compute
$$y = g^x \bmod p = 2^7 \bmod 13 = 11.$$
5. $P_B = (p, g, y) = (13, 2, 11)$ is Bob's public key.
6. $S_B = x = 7$ is Bob' private key.

Powers of Integers, Modulo 13

- 2 is a primitive root modulo 13 because for each integer $i \in Z_{13} = \{1, 2, 3, \ldots, 12\}$ there's an integer k, such that $i = 2^k \bmod 13$:

a^1	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}
1	1	1	1	1	1	1	1	1	1	1	1
2	4	8	3	6	12	11	9	5	10	7	1
3	9	1	3	9	1	3	9	1	3	9	1
4	3	12	9	10	1	4	3	12	9	10	1
5	12	8	1	5	12	8	1	5	12	8	1
6	10	8	9	2	12	7	3	5	4	11	1
7	10	5	9	11	12	6	3	8	4	2	1
8	12	5	1	8	12	5	1	8	12	5	1
9	3	1	9	3	1	9	3	1	9	3	1
10	9	12	3	4	1	10	9	12	3	4	1
11	4	5	3	7	12	2	9	8	10	6	1
12	1	12	1	12	1	12	1	12	1	12	1

Elgamal Example: Encryption

Encrypt the plaintext message $M = 3$.

- Alice gets Bob's public key $P_B = (p, g, y) = (13, 2, 11)$.
- To encrypt:
  1. Pick a random number $k = 5$:
  2. Compute:

$$\begin{aligned} a &= g^k \bmod p = 2^5 \bmod 13 = 6 \\ b &= My^k \bmod p = 3 \cdot 11^5 \bmod 13 = 8 \end{aligned}$$

- The ciphertext $C = (a, b) = (6, 8)$.

Elgamal Example: Decryption

- Bob's private key is $S_B = x = 7$.
- Bob receives the ciphertext $C = (a, b) = (6, 8)$ from Alice.
- Bob computes the plaintext M:

$$\begin{aligned} M &= b \cdot (a^x)^{-1} \bmod p \\ &= b \cdot a^{p-1-x} \bmod p \\ &= 8 \cdot 6^{13-1-7} \bmod 13 \\ &= 8 \cdot 6^5 \bmod 13 \\ &= 3 \end{aligned}$$

In-Class Exercise

- Pick the prime $p = 13$.
- Find the generator $g = 2$ for Z_{13}.
- Pick a random number $x = 9$.
- Compute
$$y = g^x \bmod p = 2^9 \bmod 13 = 5$$
- $P_B = (p, g, y) = (13, 2, 5)$ is Bob's public key.
- $S_B = x = 9$ is Bob' private key.

1. Encrypt the message $M = 11$ using the random number $k = 10$.
2. Decrypt the ciphertext from 1.

Elgamal Correctness

- Show that $M = b \cdot (a^x)^{-1} \bmod p$ decrypts.
- We have that

$$\begin{aligned} a &= g^k \bmod p \\ b &= My^k \bmod p \\ y &= g^x \bmod p \end{aligned}$$

- We get

$$\begin{aligned} b \cdot (a^x)^{-1} \bmod p &= (My^k) \cdot ((g^k)^x)^{-1} \bmod p \\ &= (My^k) \cdot (g^{kx})^{-1} \bmod p \\ &= (M((g^x)^k) \cdot (g^{kx})^{-1} \bmod p \\ &= Mg^{kx} \cdot (g^{kx})^{-1} \bmod p \\ &= Mg^{kx} \cdot g^{-kx} \bmod p \\ &= M \bmod p \\ &= M \end{aligned}$$

Elgamal Security

- The security of the scheme depends on the hardness of solving the discrete logarithm problem.
- Generally believed to be hard.

Outline

Key Exchange

- A key exchange protocol (or key agreement protocol) is a way for parties to share a secret (such as a symmetric key) over an insecure channel.
- With an active adversary (who can modify messages) we can't reliably share a secret.
- With a passive adversary (who can only eavesdrop on messages) we can share a secret.
- A passive adversary is said to be honest but curious.

Diffie-Hellman Key Exchange

- A classic key exchange protocol.
- Based on modular exponentiation.
- The secret $K_1 = K_2$ shared by Alice and Bob at the end of the protocol would typically be a shared symmetric key.

Diffie-Hellman: Algorithm

1. All parties (set-up):
   1. Pick p, a prime number.
   2. Pick g, a generator for Z_p.
2. Alice:
   1. Pick a random $x \in Z_p, x > 0$.
   2. Compute
      $$X = g^x \bmod p.$$
   3. Send X to Bob.
3. Bob:
   1. Pick a random $y \in Z_p, x > 0$.
   2. Compute
      $$Y = g^y \bmod p.$$
   3. Send Y to Alice
4. Alice computes the secret: $K_1 = Y^x \bmod p$.
5. Bob computes the secret: $K_2 = X^y \bmod p$.

Example

1. Pick $p = 13$, a prime number.
2. Pick $g = 2$, a generator for Z_{13}.
3. Alice:
   1. Pick a random $x = 3$.
   2. Compute $X = g^x \bmod p = 2^3 \bmod 13 = 8$.
4. Bob:
   1. Pick a random $y = 7$.
   2. Compute $Y = g^y \bmod p = 2^7 \bmod 13 = 11$.
5. Alice computes: $K_1 = Y^x \bmod p = 11^3 \bmod 13 = 5$.
6. Bob computes: $K_2 = X^y \bmod p = 8^7 \bmod 13 = 5$.
7. $\Rightarrow K_1 = K_2 = 5$.

In-Class Exercise

- Let $p = 19$.
- Let $g = 10$.
- Let Alice's secret $x = 7$.
- Let Bob's secret $y = 15$.

1. Compute K_1.
2. Compute K_2.

Diffie-Hellman Correctness

- Alice has computed

$$\begin{aligned} X &= g^x \bmod p \\ K_1 &= Y^x \bmod p. \end{aligned}$$

- Bob has computed

$$\begin{aligned} Y &= g^y \bmod p \\ K_2 &= X^y \bmod p. \end{aligned}$$

Diffie-Hellman Correctness...

- Alice has

$$\begin{aligned} K_1 &= Y^x \bmod p \\ &= (g^y)^x \bmod p \\ &= (g^x)^y \bmod p \\ &= X^y \bmod p \end{aligned}$$

- Bob has

$$\begin{aligned} K_2 &= X^y \bmod p \\ &= (g^x)^y \bmod p \\ &= X^y \bmod p \end{aligned}$$

- $\Rightarrow K_1 = K_2$.

Diffie-Hellman Security

- The security of the scheme depends on the hardness of solving the discrete logarithm problem.
- Generally believed to be hard.
- Diffie-Hellman Property:
  - Given
$$p, X = g^x, Y = g^y$$
  - computing
$$K = g^{xy} \bmod p$$
  is thought to be hard.

Diffie-Hellman: Man-In-The-Middle attack

1. Alice:
   1. Send $X = g^X \bmod p$ to Bob.
2. Eve:
   1. Intercept $X = g^x \bmod p$ from Alice.
   2. Pick a number t in Z_p.
   3. Send $T = g^t \bmod p$ to Bob.
3. Bob:
   1. Send $Y = g^y \bmod p$ to Alice
4. Eve:
   1. Intercept $Y = g^y \bmod p$ from Bob.
   2. Pick a number s in Z_p.
   3. Send $S = g^s \bmod p$ to Alice.

Diffie-Hellman: Man-In-The-Middle attack...

5. Alice and Eve:
   1. Compute $K_1 = g^{xS} \bmod p$
6. Bob and Eve:
   1. Compute $K_2 = g^{yT} \bmod p$

Diffie-Hellman: Man-In-The-Middle attack...

7. Alice: Send $C = E_{K_1}(M)$ to Bob
8. Eve:
   1. Intercept C.
   2. Decrypt: $M = D_{K_1}(C)$
   3. Re-encrypt: $C' = E_{K_2}(M)$
   4. Send C' to Bob
9. Bob: Send $C = E_{K_2}(M)$ to Alice
10. Eve:
    1. Intercept C.
    2. Decrypt: $M = D_{K_2}(C)$
    3. Re-encrypt: $C' = E_{K_1}(M)$
    4. Send C' to Alice.

Outline

Readings and References

- Chapter 8.1.1-8.1.5 in *Introduction to Computer Security*, by Goodrich and Tamassia.

Acknowledgments

Additional material and exercises have also been collected from these sources:

1. Igor Crk and Scott Baker, *620—Fall 2003—Basic Cryptography*.
2. Bruce Schneier, *Applied Cryptography*.
3. Pfleeger and Pfleeger, *Security in Computing*.
4. William Stallings, *Cryptography and Network Security*.
5. Bruce Schneier, *Attack Trees*, Dr. Dobb's Journal December 1999, http://www.schneier.com/paper-attacktrees-ddj-ft.html.
6. Barthe, Grégoire, Beguelin, Hedin, Heraud, Olmedo, *Verifiable Security of Cryptographic Schemes*, http://www.irisa.fr/celtique/blazy/seminar/20110204.pdf.
7. http://homes.cerias.purdue.edu/~crisn/courses/cs355_Fall_2008/lect18.pdf