CSc 466/566

Computer Security

18 : Cryptography — Public Key

Version: 2019/10/29 14:45:59

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
- Bruce Schneier facts: http://www.schneierfacts.com
- 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

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

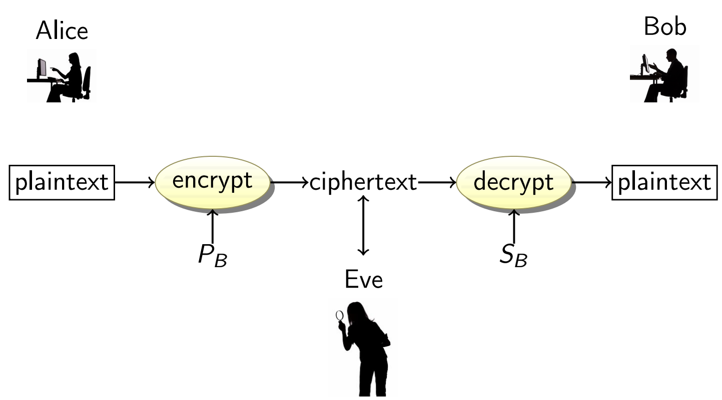
Public Key Protocol

- 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.

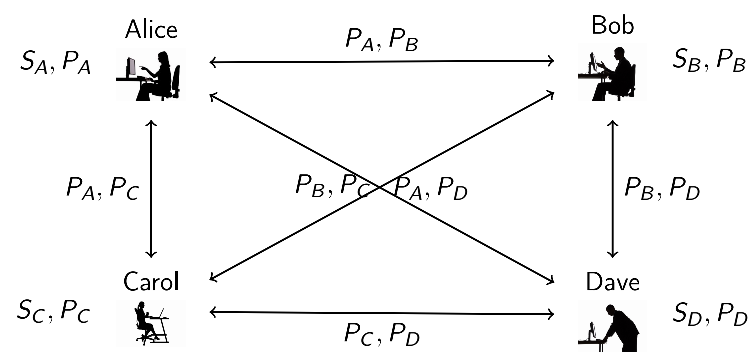
Public Key Protocol. . .

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

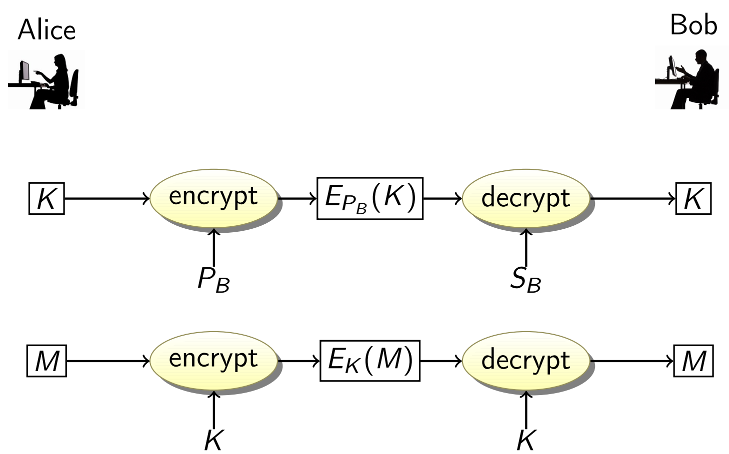
A Hybrid Protocol

- 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.

A Hybrid Protocol...

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

Modular Multiplicative Inverses

- The inverse of 4 is $\frac{1}{4}$.
- To find the inverse of x we want to compute:

$$x \cdot y = 1$$

- To compute modular multiplicative inverses we use the algorithm: GCD routine.
- The inverse of x in Z_n exists when $\mathrm{GCD}(n, x) = 1$.

Modular Multiplicative Inverses...

- Call $\mathrm{GCD}(n, x)$ which returns

$$(1, i, j)$$

such that

$$1 = ix + jn$$

- Then

$$(ix + jn) \text{ mod } n = ix \text{ mod } n = 1$$

and i is x's multiplicative inverse in Z_n.

Modular Multiplicative Inverses: Exercises

- What is $7^{-1} \pmod{11}$?
  - $GCD(7, 11) = (1, -3, 2)$
  - $(-3) \cdot 7 + (2) \cdot 11 = 1$
  - $7 * (-3) = 1 \mod 11$
  - $7 * 8 = 1 \mod 11$
  - $7^{-1} \pmod{11} = 8$

Modular Exponentiation

- Modular exponentiation:

$$x^y \text{ mod } n = \overbrace{x \cdot x \cdot \cdots \cdot x}^{y} \bmod n$$

- We compute

$$\begin{aligned} g^2 &= g \cdot g \\ g^4 &= g^2 \cdot g^2 \\ g^8 &= g^4 \cdot g^4 \end{aligned}$$

- We can then use these powers to compute g^n:

$$\begin{aligned} g^{46} &= g^{32+8+4+2} \\ &= g^{32} \cdot g^8 \cdot g^4 \cdot g^2 \end{aligned}$$

Z_n^*

- Z_n^* is the subset of Z_n of elements relatively prime with n:

$$Z_n^* = \{x \in Z_n \text{ such that } \mathrm{GCD}(x, n) = 1\}$$

- Examples:
  1. $Z_{10}^* = \{1, 3, 7, 9\}$
  2. $Z_{13}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$

Euler's Totient Function

- $\phi(n)$ is the totient of n, the number of elements of Z_n^*:

$$\phi(n) = |Z_n^*|$$

- Examples:
  1. $Z_{10}^* = \{1, 3, 7, 9\} \Rightarrow \phi(10) = 4$
  2. $Z_{13}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\} \Rightarrow \phi(13) = 12$

Outline

RSA

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

Algorithm

- Bob (Key generation):
  1. Generate two large random primes p and q.
  2. Compute $n = pq$.
  3. Compute $\phi(n) = (p-1)(q-1)$.
  4. Select a small odd integer e relatively prime with $\phi(n)$.
  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 message M for Bob):
  1. Get Bob's public key $P_B = (e, n)$.
  2. Compute $C = M^e \bmod n$.
- Bob (decrypt message C from Alice):
  1. Compute $M = C^d \bmod n$.

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.

Outline

RSA Example: Key Generation

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:

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

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.

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.

Exercise: Goodrich & Tamassia R-8.20]...

- What is the justification for Alice's advice?

✎

Exercise: Stallings pp. 270-271

Generate an RSA key-pair using $p = 17$, $q = 11$, $e = 7$.

1. $n =$

   ✎

2. $\phi(n) =$

   ✎

3. Calculate d using $de = 1 \bmod 160, d < 160$ using Euclid's algorithm. HINT:

$$\mathrm{GCD}(7, 160) = 1 = (23) \times 7 + (-1) \times 160$$

   ✎

4. $P =$

   ✎

5. $S =$

   ✎

Exercise: Stallings ...— Encrypt $M = 88$

✎

Exercise: Stallings ...— Decrypt the result!

✎

Exercise: 2012 Midterm Exam

Generate an RSA key-pair using $p = 11$, $q = 13$, $e = 7$.

1. $n =$
2. $\phi(n) =$
3. Calculate d using $de = 1 \bmod 120, d < 120$ using Euclid's algorithm. HINT:

$$\mathrm{GCD}(7, 120) = 1 = (103) \times 7 + (-1) \times 120$$

4. $P =$
5. $S =$

Exercise: 2012 Midterm Exam

- Given the RSA public key $P = (7, 65)$ and secret key $S = (29, 65)$, encrypt $M = 5$. Make sure to use an *efficient* method of computation. Show your work!

5cm

Outline

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 $M < n$

$$\begin{aligned} C^d \bmod n &= (M^e)^d \bmod n \\ &= M^{ed} \bmod n \\ &= M \end{aligned}$$

Correctness

See the book!

Outline

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.

RSA Security...

- 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.

What RSA Key Length?

- Claims:

RSA key length	Equivalent symmetric key length
1024	80
2048	112
3072	128
15360	256

- More claims:
  - 2048-bit keys are sufficient until 2030.
  - Use 3072-bit keys if you need security beyond 2030.

RSA Factoring Challenge

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

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

- This challenge was factored December 3, 2003.
- The factors are:

```
398075086424064937397125500550386491199064362
342526708406385189575946388957261768583317

472772146107435302536223071973048224632914695
302097116459852171130520711256363590397527
```

RSA Factoring Challenge...

```
Name:          RSA-640
Digits:        193
3107418240490043721350750035888567930037346022842727545720161948823
2064405180815045563468296717232867824379162728380334154710731085019
1954852900733772482278352574238645401469173660247765234660 9
```

- This challenge was factored November 2, 2005.
- The factors are:

```
16347336458092538484431338838650908598417836700330
92312181110852389333100104508151212118167511579

19008712816648221131268515739354139754718967899685
154936666385390880271038021044989571912614655 71
```

- 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
740375634795617128280467960974295731425931888892312890849362326389727
650340282662768919964196251178439958943305021275853701189680982867
331732731089309005525051168770632990723963807867100860969625379346505637963 59
```

```
Name:          RSA-768
Digits:        232
1230186684530117755130494958384962720772853569595334792197322452151726
400507263657518745202199786469389956474942774063845925192557326303453
731548268507917026122142913461670429214311602221240479274737794080665
351419597459856902143413
```

```
Name:          RSA-896
Digits:        270
412023436986659543855531365332575948179811699844327982845455626433876445
565248426198098870423161841879261420247188869492560931776375033421130
982397485150944909106910269861031862704114880866970564902903653658867
433731720813104105190864254793282601391257624033946373269391
```

```
Name:          RSA-1024
Digits:        309
135066410865995223349603216278805969938881475605667027524485143851526510
604859533833940287150571909441798207282164471551373680419703964191743
046496589274256239341020864383202110372958725762358509643110564073501
508187510676594629205563685529475213500852879416377328533906109750544334
9998111500569772368909275 63
```


RSA Factoring Challenge...

```
Name:          RSA-1536
Digits:        463
184769970321174147430683562020016440301854933866341017147178577491065169
671116124985933768430543574458561606154457179405222971773252466096064
694607124962372044202226975675668737842756238950876467844093328515749
657884341508847552829818672645133986336493190808467199043187438128336
350279547028265329780293491615581188104984490831954500984839377522725
705257859194499387007369575568843693381277961308923039256969525326162082
367649031603655137144791393234716956698806 9
```

```
Name:          RSA-2048
Digits:        617
251959084756578934940271832400483985714292821262040320277771378360436620
207075955562640185258807844069182906412495150821892985591491761845028
084891200728449926873928072877767359714183472702618963750149718246911
650776133798590957000973304597488084284017974291006424586918171951187
461215151726546322822168699875491824224336372590851418654620435767984
233871847744479207399342365848238242811981638150106748104516603773060562
016196762561338441436038339044149526344321901146575444541784240209246165
157233507787077498171257724679629263863563732899121548314381678998850
404453640235273819513786365643912120103971228221207203 57
```


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 $M_1 = M_2$!
- Use a secure padding scheme:
  Optimal asymmetric encryption padding,
  https://en.wikipedia.org/wiki/Optimal_asymmetric_encryption_padding.

How to use RSA...

- Also, side-channel attacks are possible against RSA, for example by measuring the time taken to encrypt.
- More about this later.

Outline

RSA Keys

- RSA key generation requires us to be able to generate large prime numbers, of thousand of bits.
- We can do so probabilistically.
- This step is crucial - if we don't generate *really* random primes, the security of RSA fails!

Primality Testing

- We are given an integer n and want to test if it's prime or not.
- There exists efficient methods for primality testing.
- The number of primes between 1 and n is at least $n/\ln(n)$, for $n \geq 4$.

Primality Testing

- To generate a prime number q between $n/2$ and n:
  1. Let $q \leftarrow$ a random number between $n/2$ and n;
  2. q is prime with a probability of at least $1/\ln(n)$;
  3. If $\text{isPrime}(q)$ then return q;
  4. Repeat from 1.

  We need to repeat approximately a logarithmic number of times to find a prime.

Exercise: Goodrich & Tamassia R-8.16

- Roughly how many times would you have to call a primality tester to find a prime number between 1,000,000 and 2,000,000?

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*.

Acknowledgments...

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

1. Bruce Schneier, *Attack Trees*, Dr. Dobb's Journal December 1999, http://www.schneier.com/paper-attacktrees-ddj-ft.html.
2. Barthe, Grégoire, Beguelin, Hedin, Heraud, Olmedo, *Verifiable Security of Cryptographic Schemes*, http://www.irisa.fr/celtique/blazy/seminar/20110204.pdf.
3. http://homes.cerias.purdue.edu/~crisn/courses/cs355_Fall_2008/lect18.pdf