Diffie-Hellman Key Exchange

CS6025 Data Encoding

Yizong Cheng

3-5-15

Public (Asymmetric) Key Encryption

- AES provides efficient encryption of data, when a secrete is shared by the sender and the receiver.
- Need a way to share a secrete via a open channel.
- Each side has a public key and a private key.
- A shared secrete can be generated and that can be used as the AES key.

Communication of ACM Paper

Enabling Crypto: How Radical Innovations Occur

Arnd Weber

Examining the key factors and influences in the development of cryptography.

The Libertarian Tradition in the U.S.

Public key cryptography results from the interaction of a few individuals in specific settings. The foundation of liberal communities in New England and the libertarian traditions in the U.S. have been important starting points. One of the developers is Whitfield Diffie, who grew up in a libertarian and leftist tradition: “I had come from the environment of New York City, a very left, politically active environ-

Self-Confidence

larly at MIT, I had grown up in, in which we didn't take ourselves lightly. The things talked about in conversation at MIT, the people who were talking about them took themselves seriously. If they had good ideas on them they would recognize them and work on these things."

Whitfield Diffie - Wi
en.wikipedia.org/wiki/Diffie
WIKIPEDIA
The Free Encyclopedia
Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikimedia Shop
Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia
Toolbox
Print/export
Languages
বাংলা
Deutsch
Español
Français
한국어
Italiano
עברית
Magyar
Whitfield Diffie
From Wikipedia, the free encyclopedia
(Redirected from Diffie)
"Diffie" redirects here. For the country music singer, see Joe Diffie.
Bailey Whitfield 'Whit' Diffie (born June 5, 1944) is an American cryptographer and one of the pioneers of public-key cryptography.
Diffie and Martin Hellman's paper New Directions in Cryptography was published in 1976. It introduced a radically new method of distributing cryptographic keys, that went far toward solving one of the fundamental problems of cryptography, key distribution. It has become known as Diffie–Hellman key exchange. The article also seems to have stimulated the almost immediate public development of a new class of encryption algorithms, the asymmetric key algorithms.[1]
After a long career at Sun Microsystems, where he became a Sun Fellow, Diffie served for two and a half years as Vice President for Information Security and Cryptography at the Internet Corporation for Assigned Names and Numbers (2010-2012), a visiting scholar (2009-2010) and an affiliate (2010-2012)[2] at the Freeman Spogli Institute's Center for International Security and Cooperation at Stanford University.
Contents [hide]
Whitfield Diffie
Born June 5, 1944 (age 68)
Nationality United States
Fields Cryptography
Alma mater Massachusetts Institute of Technology
Known for Diffie–Hellman key exchange
Notable awards Kanellakis Award (1976) Hamming Medal (2010)
p103-weber.pdf
Show all downloads...

Confidential Communications?

Because I had this view of cryptography in which the critical value of cryptography was that you didn't have to trust other people. And so I never understood the classical notion of a key distribution center." Diffie wondered if it were possible for all U.S. citizens to make confidential calls via telephone, a question of immediate relevance for the political movement in California.

Ralph Merkle, 1974

Independently of Diffie, Ralph Merkle, a student in Berkeley, analyzed the operation of time sharing systems "and found that quite fascinating."[2] Here, Merkle's interest in technology becomes apparent (see www.merkle.com). In 1974, he enrolled in a course on computer security engineering taught by Lance Hoffman. Merkle was analyzing how to restart secure operation of a compromised system. His starting


W Ralph Merkle - Wikip ×

en.wikipedia.org/wiki/Ralph_Merkle

Create account Log in

Article Talk Read Edit View history Search

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikimedia Shop

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia

Toolbox

Print/export

Languages
Deutsch
Français
Bahasa Indonesia

Ralph Merkle

From Wikipedia, the free encyclopedia

Ralph C. Merkle (born February 2, 1952) is one of the inventors of public key cryptography, the inventor of cryptographic hashing, and more recently a researcher and speaker on molecular nanotechnology and cryonics. Merkle appears in the science fiction novel *The Diamond Age*, involving nanotechnology.

Ralph Merkle	
Ralph Merkle at the Singularity Summit 2007	
Born	February 2, 1952 (age 61)
Citizenship	American
Nationality	American
Fields	Public key cryptography, Molecular nanotechnology, and Cryonics
Institutions	Singularity University Alcor Life Extension Foundation Institute for Molecular

Contents [hide]

Early life and education [edit]

Merkle graduated from Livermore High School in 1970 and proceeded to study computer science

p103-weber.pdf

Show all downloads...

Merkle’s Puzzle System, 1974

Merkle’s Puzzle System

Overview

Merkle’s approach is to communicate a cryptographic key from one person to another by hiding it in a large collection of puzzles. For example, say Alice manufactures a million or more puzzles and sends them over an exposed communication channel to Bob. Each puzzle contains a cryptographic key in a recognizable standard format. When Bob receives the puzzles, he picks one and solves it. In order to inform Alice which puzzle he has solved, Bob uses the key it contains to encrypt a fixed test message, which he transmits to Alice. Alice now tries her million keys on the test message until she finds the one that works. The task facing an intruder is more arduous. Rather than selecting one of the puzzles to solve, an intruder must solve on average half of them. [1]

First Concept*

“The method as I first conceived it was: generate roughly k random numbers from a space of about k^2 possible numbers. Apply a one-way hash function to these k random numbers and transmit them from A to B. B then generates random numbers from the same space of k^2 possible numbers and applies the same one-way hash function until there is a ‘collision’, that is, B accidentally generates one of the same random numbers that A generated. The probability of such a collision is surprisingly high (check the Birthday Paradox). B then encrypts further messages with the common random number. A, to determine which random number is correct, tries all k random numbers that it generated on some test message encrypted by B.”

*Merkle in correspondence with the author—see [7].

Rejection Comments, 1975

- Thank you very kindly for your communication of October 7, with the enclosed paper on Secure communications over insecure channels. I am sorry to have to inform you that the paper is not in the mainstream of present cryptography thinking and I would not recommend that it be published in the Communications of the ACM for the following reasons: The paper proposes to describe cryptographic security by transmitting under various unrealistic working assumptions, puzzles conveying key information, a puzzle which is just another word to talk about a cryptosystem. The strength of the system hinges strongly on the quality of the puzzle transformation, these are not defined.
- Experience shows that it is extremely dangerous to transmit key information in the clear. Such practices of the legitimate user open the setup to illegitimate test procedures which only a very strong system could resist.

Diffie explained his approach to Hellman, who was in contact with Jim Massey of the IEEE *Transactions on Information Theory*, and they prepared a paper for submission to that journal. A preversion reached Merkle's friend Blatman. Thus, Merkle and Diffie got to know each other. Merkle approached Diffie and Hellman in order to escape from the loneliness of his efforts. Diffie, Hellman, and Merkle then discussed the capabilities and problems of the public key approach. In 1976, the Diffie-Hellman paper was published [2]. Merkle's approach was published in 1978 as "Secure Communications Over Insecure Channels" in *Communications of the ACM* [8]; at the time of publication, the concept had been established.

Martin Hellman - Wi
en.wikipedia.org/wiki/Martin_Hellman

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikimedia Shop

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia

Toolbox

Print/export

Languages
Česky
Deutsch
Español
Français
Italiano
עברית
Magyar
日本語

Martin Hellman

From Wikipedia, the free encyclopedia

Martin Edward Hellman (born October 2, 1945) is an American cryptologist, and is best known for his invention of public key cryptography in cooperation with Whitfield Diffie and Ralph Merkle.[1][2][3][4][5][6][7] Hellman is a long-time contributor to the computer privacy debate and is more recently known for promoting risk analysis studies on nuclear threats, including the NuclearRisk.org[8] website.

Martin Edward Hellman

Born	October 2, 1945 (age 67) New York
Nationality	American
Fields	Cryptography
Alma mater	New York University (BSc) Stanford University (PhD)
Thesis	*Learning with Finite Memory* (1969)
Doctoral advisor	Thomas Cover
Known for	Diffie–Hellman key exchange

Website
www-ee.stanford.edu/~hellman

Contents [hide]

Early life [edit]

Hellman graduated from the Bronx High School of Science. He went on to earn his Bachelor's degree from New York University in 1966, and at

p103-weber.pdf
Show all downloads...

Diffie-Hellman Key Exchange

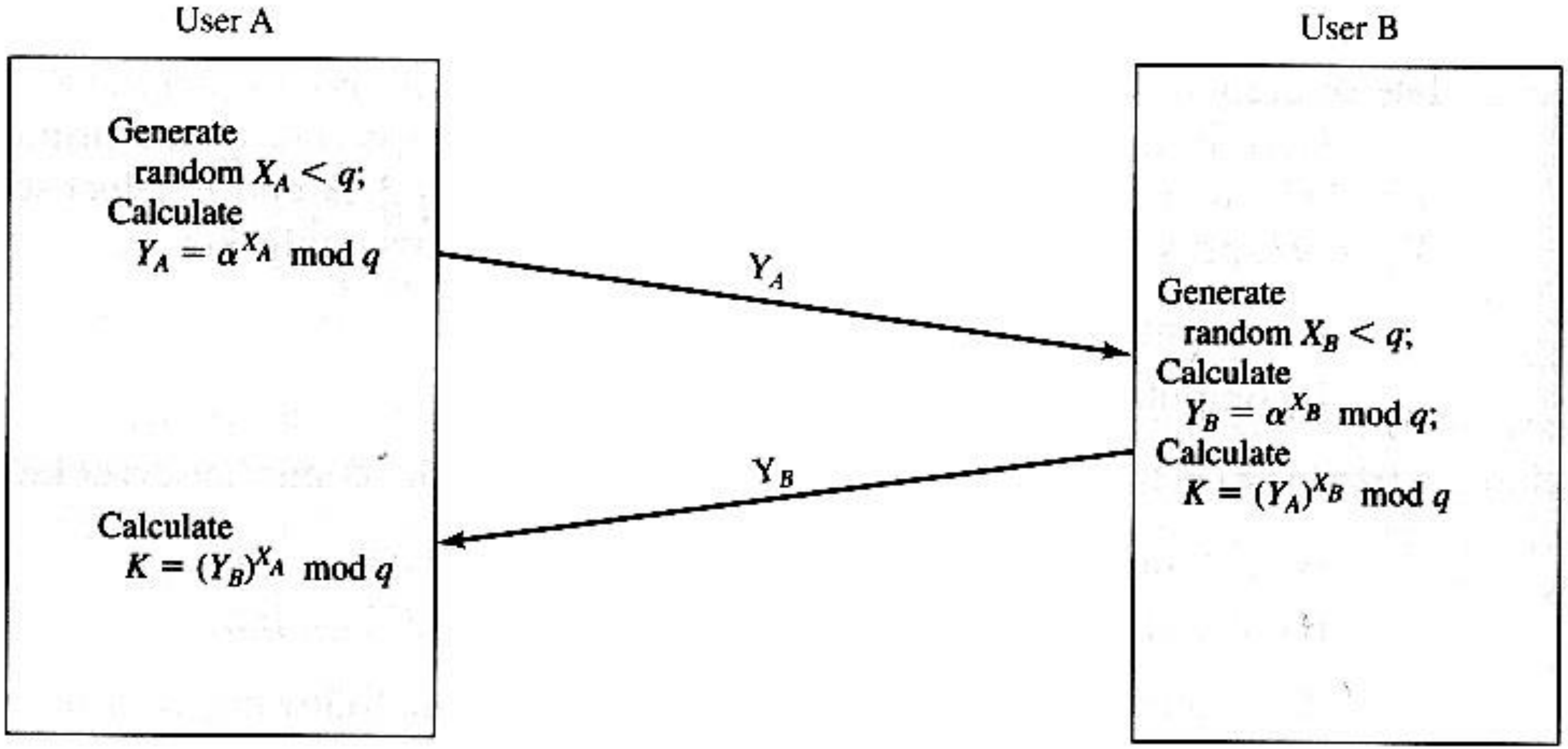


Figure 10.2 Diffie-Hellman Key Exchange

Global Public Elements

q	prime number
α	$\alpha < q$ and α a primitive root of q

User A Key Generation

Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{XA} \bmod q$

User B Key Generation

Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{XB} \bmod q$

Calculation of Secret Key by User A

$K = (Y_B)^{XA} \bmod q$

Calculation of Secret Key by User B

$K = (Y_A)^{XB} \bmod q$

Figure 10.1 The Diffie-Hellman Key Exchange Algorithm

K is the Shared Secret

$$
\begin{aligned}
K &= (Y_B)^{X_A} \bmod q \\
&= (\alpha^{X_B} \bmod q)^{X_A} \bmod q \\
&= (\alpha^{X_B})^{X_A} \bmod q \\
&= \alpha^{X_B X_A} \bmod q \\
&= (\alpha^{X_A})^{X_B} \bmod q \\
&= (\alpha^{X_A} \bmod q)^{X_B} \bmod q \\
&= (Y_A)^{X_B} \bmod q
\end{aligned}
$$

Table 8.1 Primes Under 2000

2	101	211	307	401	503	601	701	809	907	1009	1103	1201	1301	1409	1511	1601	1709	1801	1901
3	103	223	311	409	509	607	709	811	911	1013	1109	1213	1303	1423	1523	1607	1721	1811	1907
5	107	227	313	419	521	613	719	821	919	1019	1117	1217	1307	1427	1531	1609	1723	1823	1913
7	109	229	317	421	523	617	727	823	929	1021	1123	1223	1319	1429	1543	1613	1733	1831	1931
11	113	233	331	431	541	619	733	827	937	1031	1129	1229	1321	1433	1549	1619	1741	1847	1933
13	127	239	337	433	547	631	739	829	941	1033	1151	1231	1327	1439	1553	1621	1747	1861	1949
17	131	241	347	439	557	641	743	839	947	1039	1153	1237	1361	1447	1559	1627	1753	1867	1951
19	137	251	349	443	563	643	751	853	953	1049	1163	1249	1367	1451	1567	1637	1759	1871	1973
23	139	257	353	449	569	647	757	857	967	1051	1171	1259	1373	1453	1571	1657	1777	1873	1979
29	149	263	359	457	571	653	761	859	971	1061	1181	1277	1381	1459	1579	1663	1783	1877	1987
31	151	269	367	461	577	659	769	863	977	1063	1187	1279	1399	1471	1583	1667	1787	1879	1993
37	157	271	373	463	587	661	773	877	983	1069	1193	1283		1481	1597	1669	1789	1889	1997
41	163	277	379	467	593	673	787	881	991	1087		1289		1483		1693			1999
43	167	281	383	479	599	677	797	883	997	1091		1291		1487		1697			
47	173	283	389	487		683		887		1093		1297		1489		1699			
53	179	293	397	491		691				1097				1493					
59	181			499										1499					
61	191																		
67	193																		
71	197																		
73	199																		
79																			
83																			
89																			
97																			

Table 8.3 Powers of Integers, Modulo 19

a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{14}	a^{15}	a^{16}	a^{17}	a^{18}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1
3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	1
4	16	7	9	17	11	6	5	1	4	16	7	9	17	11	6	5	1
5	6	11	17	9	7	16	4	1	5	6	11	17	9	7	16	4	1
6	17	7	4	5	11	9	16	1	6	17	7	4	5	11	9	16	1
7	11	1	7	11	1	7	11	1	7	11	1	7	11	1	7	11	1
8	7	18	11	12	1	8	7	18	11	12	1	8	7	18	11	12	1
9	5	7	6	16	11	4	17	1	9	5	7	6	16	11	4	17	1
10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	1
11	7	1	11	7	1	11	7	1	11	7	1	11	7	1	11	7	1
12	11	18	7	8	1	12	11	18	7	8	1	12	11	18	7	8	1
13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	1
14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	1
15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	1
16	9	11	5	4	7	17	6	1	16	9	11	5	4	7	17	6	1
17	4	11	16	6	7	5	9	1	17	4	11	16	6	7	5	9	1
18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1

Table 4.5 Arithmetic in GF(7)

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

(a) Addition modulo 7

×	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

(b) Multiplication modulo 7

w	$-w$	w^{-1}
0	0	—
1	6	1
2	5	4
3	4	5
4	3	2
5	2	3
6	1	6

(c) Additive and multiplicative inverses modulo 7

Multiplication and Exponentiation Tables for GF(2^3)

- 1 2 3 4 5 6 7 1 1 1 1 1 1 1
- 2 4 6 3 1 7 5 2 4 3 6 7 5 1
- 3 6 5 7 4 1 2 3 5 4 7 2 6 1
- 4 3 7 6 2 5 1 4 6 5 2 3 7 1
- 5 1 4 2 7 3 6 5 7 6 3 4 2 1
- 6 7 1 5 3 2 4 6 2 7 4 5 3 1
- 7 5 2 1 6 4 3 7 3 2 5 6 4 1
- Question: Why is $a^7 = 1$ for all nonzero a?

The Multiplication Table of a Finite Field

- 1 2 3 4 5 6 7 (1*1)(1*2)(1*3)(1*4)(1*5)(1*6)(1*7)
- 2 4 6 3 1 7 5 (2*1)(2*2)(2*3)(2*4)(2*5)(2*6)(2*7)
- 3 6 5 7 4 1 2 (3*1)(3*2)(3*3)(3*4)(3*5)(3*6)(3*7)
- 4 3 7 6 2 5 1 (4*1)(4*2)(4*3)(4*4)(4*5)(4*6)(4*7)
- 5 1 4 2 7 3 6 (5*1)(5*2)(5*3)(5*4)(5*5)(5*6)(5*7)
- 6 7 1 5 3 2 4 (6*1)(6*2)(6*3)(6*4)(6*5)(6*6)(6*7)
- 7 5 2 1 6 4 3 (7*1)(7*2)(7*3)(7*4)(7*5)(7*6)(7*7)
- The product of each row is the same as
- 1*2*3*4*5*6*7 = (a*1)(a*2)(a*3)(a*4)(a*5)(a*6)(a*7) =
- a^7(1*2*3*4*5*6*7) ➔ a^7 = 1 after multiplying inverse of 1*2*3*4*5*6*7 .

Fermat's Theorem

- In general, when we have a (q-1)x(q-1) multiplication table for GF(q) with each row a permutation of q-1 elements, we have the Fermat's Theorem:
- For any of the q-1 elements, a, we have $a^{q-1} = 1$.
- If p is a prime number, then for all $a > 0$ and $a < p$, we have $a^{p-1} = 1 \bmod p$ (Z_p is GF(p)).

Fermat's Theorem as Primality Test

- If n is prime, then $a^{n-1} \bmod n = 1$.
- $a^{(n-1)/2} \bmod n$ is either 1 or $n - 1$.
- If $a^{(n-1)/2} = 1$ then $a^{(n-1)/4}$ is 1 or $n - 1$.
- Let $n – 1 = 2^k q$ where q is odd.
- If a^q is not 1 or n-1 and $a^{2q}, a^{4q},…, a^{(k-1)q}$ are not n-1, then a^{n-1} is not 1 and n is not prime.
- Otherwise, n may still not be prime.

Miller-Rabin Primality Test

- However, the probability for an a to pass the test when n is not prime is smaller than ¼.
- If we randomly choose a for the test t times, and all passed, then the probability that n is not prime is less than 4^{-t}.
- java's BigInteger has constructors and isProbablePrime(t/2).

Primitive Elements in Z_p

- 1,2,…,p-1 form a cyclic multiplicative group in Z_p.
- Any generator of the group is called a primitive element of Z_p, or a primitive root of p.
- 2,3,10,13,14,15 are primitive elements of Z_{19}. Or primitive roots of 19.

Table 8.4 Tables of Discrete Logarithms, Modulo 19

(a) Discrete logarithms to the base 2, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$log_{2,19}(a)$	18	1	13	2	16	14	6	3	8	17	12	15	5	7	11	4	10	9

(b) Discrete logarithms to the base 3, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$log_{3,19}(a)$	18	7	1	14	4	8	6	3	2	11	12	15	17	13	5	10	16	9

(c) Discrete logarithms to the base 10, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$log_{10,19}(a)$	18	17	5	16	2	4	12	15	10	1	6	3	13	11	7	14	8	9

(d) Discrete logarithms to the base 13, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$log_{13,19}(a)$	18	11	17	4	14	10	12	15	16	7	6	3	1	5	13	8	2	9

(e) Discrete logarithms to the base 14, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$log_{14,19}(a)$	18	13	7	8	10	2	6	3	14	5	12	15	11	1	17	16	4	9

(f) Discrete logarithms to the base 15, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$log_{15,19}(a)$	18	5	11	10	8	16	12	15	4	13	6	3	7	17	1	2	14	9

Diffie-Hellman Key Exchange

- Everybody knows q and α.
- Each select a *private key* X < q and publish a *public key* Y = α^{X} mod q
- Now we have Xa and Ya = α^{Xa} mod q for user A and Xb and Yb = α^{Xb} mod q for user B.
- A gets K = $(Yb)^{Xa}$ mod q = α^{XbXa} mod q
- B gets K = $(Ya)^{Xb}$ mod q = α^{XaXb} mod q
- Published by Diffie, Hellman in 1976 but discovered before by British intelligence.

A General View of Diffie-Hellman

- Addition in Z_q is not used.
- Multiplication in Z_q makes {1, 2, …, q – 1} a finite *cyclic group* with a *generating element* α, which is also called a primitive element of Z_q .
- Works for any cyclic group.

Modular Arithmetic in Z_q

- Addition and multiplication in Z_q can be executed as integer operations followed by mod q (divide by q and take the remainder, which is between 0 and q – 1).
- Subtraction by d is addition with q – d.
- Division by d is multiplication by d^{-1}, or the multiplicative inverse of d mod q.
  - $(a + b) \bmod q = ((a \bmod q) + (b \bmod q)) \bmod q$
  - $(a * b) \bmod q = ((a \bmod q) + (b \bmod q)) \bmod q$
  - $(a^b)^c \bmod q = (a^b \bmod q)^c \bmod q$

Discrete Logarithm is Hard

- To break Diffie-Hellman, one needs to find the private key from the public key.
- Diffie-Hellman is secure because knowing $Y = \alpha^X \bmod q$, α, and q, it is computationally infeasible to find X.
- When q is a prime number and α is a *primitive element* in $GF(q) = Z_q$, X is the *discrete logarithm* of Y base α in Z_q.
- Wanted: a large prime number q and a primitive element of Z_q, α (called a *primitive root* of q).

How to Find a Primitive Element in Z_q?

- The last column of the exponentiation table is all 1.
- Any element that is NOT primitive has an 1 at a position that is not the last column.
- This position has to be a divisor if q-1.
- If (q-1)/2 is also a prime, to test if α is a primitive element, we only have to test $\alpha^{(q-1)/2}$ to make sure that it is not 1.
  - Maybe in this case, any element can play α.

Internet Key Exchange (IKE)

- Standardizes secure socket layer (SSL) parameters (https uses this).
- Diffie-Hellman group 1 (default) 768-bit
- group 2 1024-bit
- group 5 1536-bit

Diffie-Hellman Group 1

- Uses p below and α = 2.

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE65381
FFFFFFFF FFFFFFFF
```

Diffie-Hellman Group 5

1536-bit MODP Group

The prime is: 2^1536 - 2^1472 - 1 + 2^64 * { [2^1406 pi] + 741804 }

Its hexadecimal value is:

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA237327 FFFFFFFF FFFFFFFF
```

The generator is: 2.

Java BigInteger

- Random rand = new Random();
- BigInteger pp = new BigInteger(100, 200, rand);
- BigInteger p1 = pp.subtract(BigInteger.ONE).divide(new BigInteger(“2”));
- if (p1.isProbablePrime(200)) …

Other Methods of BigInteger

- Constructors BigInteger(int, Random) and BigInteger(String, int)
- int compareTo(BigInteger)
- BigInteger modInverse(BigInteger)
- BigInteger modPow(BigInteger, BigInteger)
- BigInteger multiply(BigInteger)
- BigInteger subtract(BigInteger)
- String toString(int)
- boolean testBit(int) and BigInteger setBit(int)

Man-in-the-Middle Attack

1. Darth prepares for the attack by generating two random private keys X_{D1} and X_{D2} and then computing the corresponding public keys Y_{D1} and Y_{D2}.
2. Alice transmits Y_A to Bob.
3. Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates $K2 = (Y_A)^{X_{D2}} \bmod q$.
4. Bob receives Y_{D1} and calculates $K1 = (Y_{D1})^{X_B} \bmod q$.
5. Bob transmits Y_B to Alice.
6. Darth intercepts Y_B and transmits Y_{D2} to Alice. Darth calculates $K1 = (Y_B)^{X_{D1}} \bmod q$.
7. Alice receives Y_{D2} and calculates $K2 = (Y_{D2})^{X_A} \bmod q$.

1. Alice sends an encrypted message M: $\mathrm{E}(K2, M)$.
2. Darth intercepts the encrypted message and decrypts it to recover M.
3. Darth sends Bob $E(K1, M)$ or $\mathrm{E}(K1, M')$, where M' is any message. In the first case, Darth simply wants to eavesdrop on the communication without altering it. In the second case, Darth wants to modify the message going to Bob.

Homework 14: due 3-11-15

- H14A.java is a UDP server that generates its Diffie-Hellman key pair and waits for a client. Group 5 parameters are used.
- H14B.java is a UDP client that generates its Diffie-Hellman key pair and sends its public key as a UDP datagram to the server and then waits for the reply from the server.
- The server (H14A) receives the client's public key and sends its public key to the client.
- After receiving each other's public key, both computes a shared secret part of which may be used as the symmetric key for AES encryption.
- Complete these programs and demonstrate the operation in two command windows on one or two computers.
- Write H14C.java to read the Group 5 parameters (q and alpha/a) and show that both q and (q-1)/2 are prime and a is a primitive root of q.

The UDP Server H14A

```
import java.io.*;
import java.util.*;
import java.math.*;
import java.net.*;

public class H14A{
  static int MAXBF = 1024;
  String hexQ = null;
  BigInteger q = null;
  static BigInteger alpha = new BigInteger("2");
  BigInteger privateKey;
  BigInteger publicKey;
  BigInteger clientPublicKey;
  byte[] publicKeyBytes = null;
  BigInteger preMasterSecret;
  String hexkey = null;
```

```java
void readQ(String filename){
    Scanner in = null;
    try {
     in = new Scanner(new File(filename));
    } catch (FileNotFoundException e){
      System.err.println(filename + " not found");
      System.exit(1);
    }
    hexQ = in.nextLine();
    in.close();
    q = new BigInteger(hexQ, 16);
  }

 void generateKeyPair(){
   Random random = new Random();
   privateKey = new BigInteger(1235, random);
   publicKey = // what is the public key?
   publicKeyBytes = publicKey.toByteArray();
 }
```

```java
 void runUDPServer(int serverPort){
   DatagramSocket ds = null;
   DatagramPacket dp = null;
   byte[] buff = new byte[MAXBF];
   try {
     ds = new DatagramSocket(serverPort);
     dp = new DatagramPacket(buff, MAXBF);
     ds.receive(dp); // blocking until receiving
     int len = dp.getLength();
     byte[] clientPublicKeyBytes = new byte[len];
     for (int i = 0; i < len; i++) clientPublicKeyBytes[i] = buff[i];
     clientPublicKey = new BigInteger(clientPublicKeyBytes);
     InetAddress iadd = dp.getAddress(); // client's IP address
     int clientPort = dp.getPort();
     System.out.println(" from " + iadd.getHostAddress() + ":" + clientPort);
     dp = new DatagramPacket(publicKeyBytes, publicKeyBytes.length, iadd, clientPort);
     ds.send(dp);

   } catch (IOException e){
     System.err.println("IOException");
     return;
   }
}
```

Server’s main

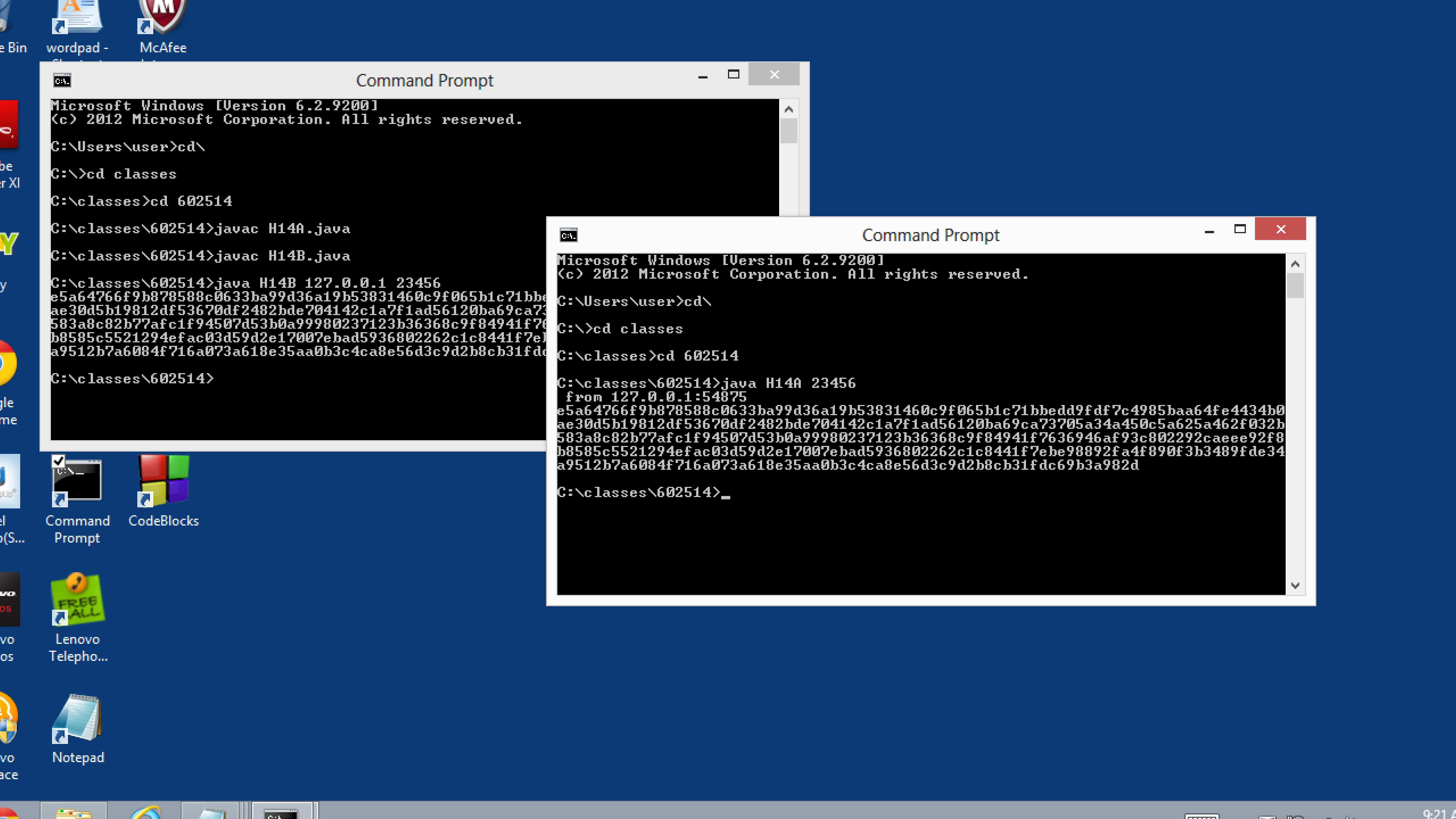
```
void computeSharedSecret(){
   preMasterSecret = // how to get the shared secret with Diffie-Hellman?
   hexkey = preMasterSecret.toString(16);
   System.out.println(hexkey);
}

public static void main(String[] args){
  if (args.length < 1){
    System.err.println("Usage: java H14A port");
    System.exit(1);
  }
  H14A h14 = new H14A();
  h14.readQ("DHgroup5.txt");
  h14.generateKeyPair();
  h14.runUDPServer(Integer.parseInt(args[0]));
  h14.computeSharedSecret();
}
```

```java
public class H14B{  // The client
  static int MAXBF = 1024;
  String hexQ = null;
  BigInteger q = null;
  static BigInteger alpha = new BigInteger("2");
  BigInteger privateKey;
  BigInteger publicKey;
  BigInteger serverPublicKey;
  byte[] publicKeyBytes = null;
  BigInteger preMasterSecret;
  String hexkey = null;

 public static void main(String[] args){
   if (args.length < 2){
     System.err.println("Usage: java H14B serverIP serverPort");
     System.exit(1);
   }
   H14B h14 = new H14B();
   h14.readQ("DHgroup5.txt");
   h14.generateKeyPair();
   h14.runUDPClient(args[0], Integer.parseInt(args[1]));
   h14.computeSharedSecret();
 }
```

```
void runUDPClient(String serverIP, int serverPort){
  InetAddress iadd = null;
  try {
    iadd = InetAddress.getByName(serverIP);
  } catch (UnknownHostException e){  System.err.println("Exception"); return; }
  DatagramSocket ds = null;
  DatagramPacket dp = null;
  byte[] buff = new byte[MAXBF];
  try {
    ds = new DatagramSocket();
    dp = new DatagramPacket(publicKeyBytes, publicKeyBytes.length, iadd, serverPort);
    ds.send(dp);
    dp = new DatagramPacket(buff, MAXBF);
    ds.receive(dp);  // blocking until receiving
    int len = dp.getLength();
    byte[] serverPublicKeyBytes = new byte[len];
    for (int i = 0; i < len; i++) serverPublicKeyBytes[i] = buff[i];
    serverPublicKey = new BigInteger(serverPublicKeyBytes);
  } catch (IOException e){
    System.err.println("IOException");
    return;
  }
}
```


wordpad -
McAfee
Command Prompt
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.
C:\Users\user>cd\
C:\>cd classes
C:\classes>cd 602514
C:\classes\602514>javac H14A.java
C:\classes\602514>javac H14B.java
C:\classes\602514>java H14B 127.0.0.1 23456
e5a64766f9b878588c0633ba99d36a19b53831460c9f065b1c71bbe
ae30d5b19812df53670df2482bde704142c1a7f1ad56120ba69ca73
583a8c82b77afc1f94507d53b0a99980237123b36368c9f84941f76
b8585c5521294efac03d59d2e17007ebad5936802262c1c8441f7el
a9512b7a6084f716a073a618e35aa0b3c4ca8e56d3c9d2b8cb31fdc
C:\classes\602514>
Command Prompt
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.
C:\Users\user>cd\
C:\>cd classes
C:\classes>cd 602514
C:\classes\602514>java H14A 23456
from 127.0.0.1:54875
e5a64766f9b878588c0633ba99d36a19b53831460c9f065b1c71bbedd9fdf7c4985baa64fe4434b0
ae30d5b19812df53670df2482bde704142c1a7f1ad56120ba69ca73705a34a450c5a625a462f032b
583a8c82b77afc1f94507d53b0a99980237123b36368c9f84941f7636946af93c802292caeee92f8
b8585c5521294efac03d59d2e17007ebad5936802262c1c8441f7ebe98892fa4f890f3b3489fde34
a9512b7a6084f716a073a618e35aa0b3c4ca8e56d3c9d2b8cb31fdc69b3a982d
C:\classes\602514>_
Command Prompt
CodeBlocks
Lenovo Telepho...
Notepad

H14C.java

```
void testPrimality(){
  if (q.isProbablePrime(200))
   System.out.println("q is probably prime");
  p = // your code for (q-1)/2
  if (p.isProbablePrime(200))
   System.out.println("p is probably prime");
}

void testPrimitiveness(){
  BigInteger 2pq = // compute pow(2, p) mod q
  System.out.println(2pq.toString(16));
}
```