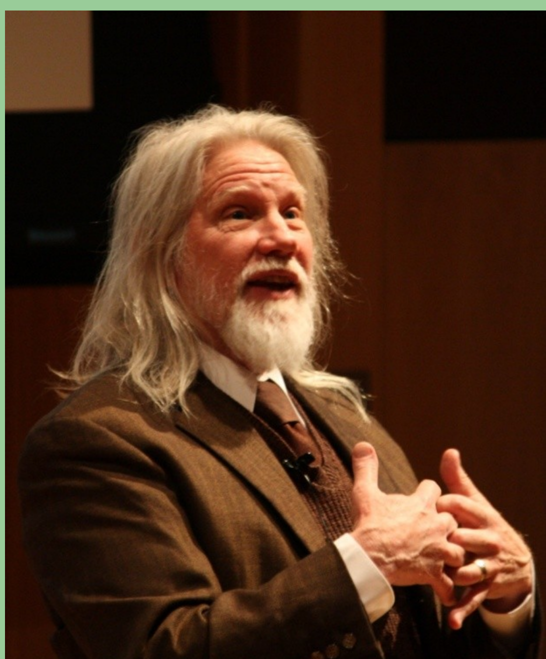
Public Key Cryptography



Inside PKCS

Rajaram Pejaver

Press the <Page Down> key to advance
Press the <Back space> key to replay a slide

Outline

This is *not* a talk about PKI. This is a prerequisite for PKI.

- Background: Shared Secret Keys
- Public Key Cryptography
  - Key Exchange explained
- Key Exchange using Diffie-Hellman
- Key Exchange using RSA
- Signatures using RSA
- Applications of PKC
- Problems/issues with PKC

Background: Shared Secret Keys

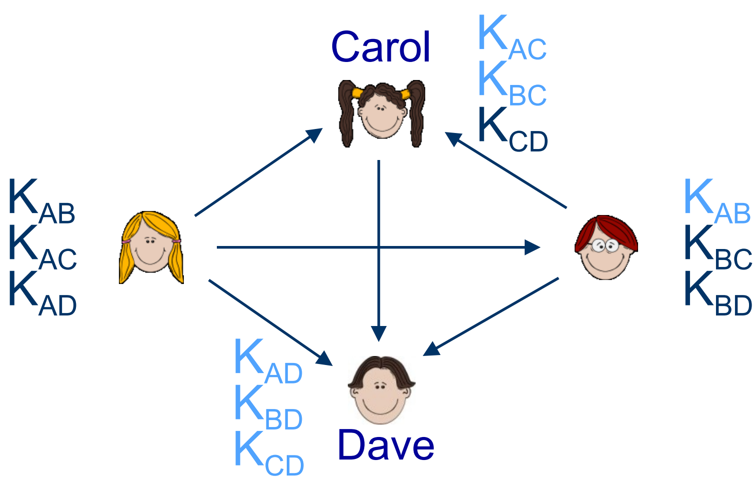
It takes two to share a secret

Say, Alice & Bob communicate using shared secret keys

- Alice encrypts text and Bob decrypts it, using the same key

Only 1 Key is needed.

- But when Alice, Bob, Carol & Dave want to communicate



Total of 6 Keys needed.

$$N_{keys} = n * (n - 1) / 2$$

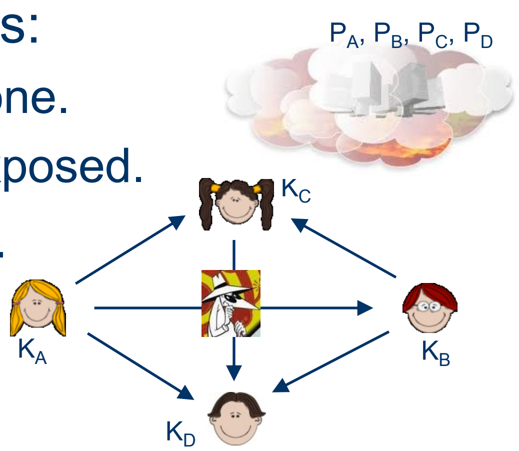
For 100 communicants, N_{keys} will be ~5000 !

For 1000 communicants, N_{keys} will be ~500,000 !!

Public Key Cryptography

No sharing of private information.

- Public Key Cryptography keys come in pairs:
  - Public component: P_A, Shared with everyone.
  - Private component: K_A, Never shared or exposed.
- Each person needs only one key pair, ever.
  - Scales linearly: 4 keys for 4 subjects.
  - Much easier to secure private component.



- Problem: Encryption is slow & compute intensive.
  - Cannot encrypt messages with PKC.
- Solution: Use PKC to establish a shared secret session key.
  - This is called Key Exchange.
  - Alice & Bob agree to use K_{AB} without anyone else finding out.

Diffie-Hellman Algorithm (1)

Key Exchange Only. No direct encryption. No signatures.

- First, everyone agrees on a number for 'generator value': g
- Each person picks a random number as Private Key: K_A
- Each person computes their Public Key, P_A: g^{K_A} (i.e. g ^ K_A)
- Alice and Bob exchange their public keys. $P_A \leftarrow\rightarrow P_B$
- Each person exponentiates other's public key with their own private key.

	Private	Public Key	Exponentiation
Alice	K_A	$P_A = g$ ^ K_A	P_A ^ K_B = (g ^ K_A) ^ K_B = g ^ ($K_A * K_B$)
Bob	K_B	$P_B = g$ ^ K_B	P_B ^ K_A = (g ^ K_B) ^ K_A = g ^ ($K_A * K_B$)

- Ta da… Both parties have computed the same value: g ^ ($K_A * K_B$)
  - They can use this value to compute a shared secret key .

Diffie-Hellman Algorithm (2)

Example: using regular math and small numbers.

- First, everyone agrees on a number for 'generator value': ***g = 8***
- Each person picks a random number as Private Key:
  - $K_A = 6$, $K_B = 4$
- Each person computes their Public Key, P_A and P_B:
  - $P_A = g^{K_A} = 8^6 = 262\,144$, $P_B = g^{K_B} = 8^4 = 4\,096$
- Each person exponentiates other's public key with their own private key.

	Private	Public Key	Exponentiation
Alice	***6***	***262 144***	***262 144^4 = 4 722 366 482 869 645 213 696***
Bob	***4***	***4 096***	***4 096^6 = 4 722 366 482 869 645 213 696***

- They can use this value to compute a shared secret key.

Diffie-Hellman Algorithm (3)

Some analysis.

- Note how big the result got, even though we used single digit keys.
  - The result was 22 digits long!
- Yet, Alice's private key can be easily hacked.
  - Given the value of ***g*** and Alice's public key, calculate the log function:

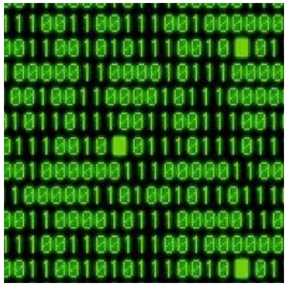
    $K_A = \log_{\boldsymbol{g}}(P_A)$

- One solution, we could use much larger numbers for keys and for ***g.***
  - That helps a bit, but not anywhere near enough.
- Next solution: Use a different number system: Modulo arithmetic field.
  - It is much harder to calculate log functions in modulo fields.
    - Discrete logarithm problem in modular fields is NP complete.
  - In addition to ***g***, we need to pick a system wide prime modulus ***p***.
    - ***p*** > ***g***

Diffie-Hellman Algorithm (4)

Using modular arithmetic and small numbers.

- Besides ***g***, agree on ‘prime modulus’ ***p***: $g = 8, \; p = 17$
- Each person picks a random number as Private Key:
  - $K_A = 6,$ $K_B = 4$



- Each person computes their Public Key, P_A and P_B:
  - $P_A = g^{K_A} \% 17 = 8^6 \% 17 = 262\,144 \% 17 = 4$
  - $P_B = g^{K_B} \% 17 = 8^4 \% 17 = 4\,096 \% 17 = 16$
- Each person exponentiates other’s public key with their own private key.

	Private	Public Key	Exponentiation
Alice	***6***	***4***	***4 ^ 4 % 17 = 256 % 17 = 1***
Bob	***4***	***16***	***16 ^ 6 % 17 = 16 777 216 % 17 = 1***

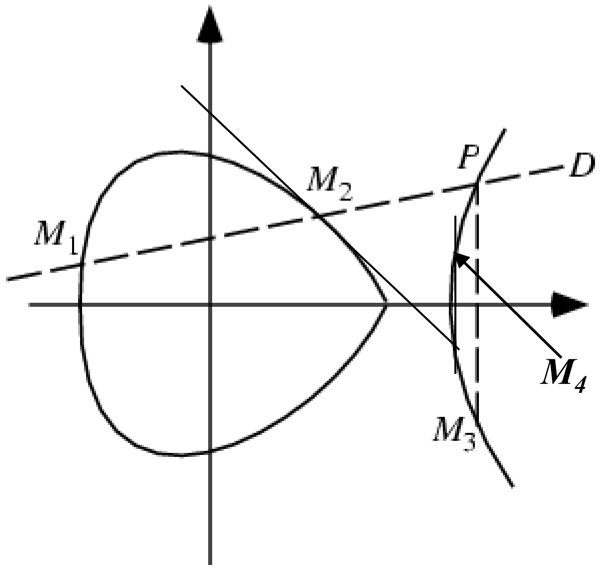
- Foundation: Given ***g***, ***p*** and P_A (g^$K_A \% p$) it is not easy to calculate K_A.

Diffie-Hellman Algorithm (5)

Elliptic Curve math.

- Given a polynomial of the form: $y^2 = x^3 + ax + b$
  - The points on the curve form a closed set of numbers constituting a field.
  - Adding two points $M_3 = M_1 + M_2 = -P$
  - Doubling a point $M_4 = M_2 + M_2$
  - Multiply two points $M_5 = M_1 * M_2 = M_1 + M_1 + M_1 \ldots$ (M_2 times)
- Foundation: Given M_5 and M_1, it is not easy to g
  - Can't easily find the multiplicative inverse
- Multiply other's public key with own private key
  - Alice and Bob have both calculated: $g * K_A * K_B$

		Public Key	Multiplication
Alice	K_A	$P_A = g * K_A$	$P_A * K_B = g * K_A * K_B$
Bob	K_B	$P_B = g * K_B$	$P_B * K_A = g * K_B * K_A$



Diffie-Hellman Algorithm (6)

EC Cryptography: Final thoughts.

- EC Math is more complex & slower, hence smaller keys are adequate.
  - Further, EC performance scales better.
- Key sizes (in bits) for comparable strengths in different systems
  - Cost factor compares EC to PKC

Symmetric Keys	PKC Keys	EC Keys	Cost Factor
80	1024	160	3
112	2048	224	6
128	3072	256	10
192	7680	384	32
256	15360	521	64

- Remember: Diffie Hellman supports
  - ONLY Key Exchange.
  - No signatures.
  - No direct encryption.

Key Exchange using RSA (1)

Basic Concepts.

- Key generation:
  - Select modulus ***n***, product of two primes: $n = p * q$
  - Select public exponent ***e***.
    - A good choice is F_4 (65537).
  - Select private exponent d
    - such that $d * e \equiv 1$ modulo **LCM**($p - 1$, $q - 1$)
    - ***d*** is the multiplicative inverse of ***e***
- Encryption consists of modular exponentiation of plaintext ***m*** with ***e***
  - $m^e \% n \rightarrow c$ (where $m < n$)
- Decryption consists of modular exponentiation of encrypted text ***c*** with ***d***
  - $c^d \% n \rightarrow (m^e)^d \% n \rightarrow m^{ed} \% n \rightarrow m$
  - Because in the exponent, $e*d = 1$ (kind of ☺)

Key Exchange using RSA (2)

Example: using modulo math and small numbers.

- Key generation by Alice:
  - ***p*** = 971, ***q*** = 719, ***n*** = 698149 (***p********q***)
  - ***e*** = 3, ***d*** = 464307
- Encryption by Bob with Public Key ***e***: Plain text m = 123
  - 123^3 % 698149 = 464569
- Decryption by Alice with Private Key ***d***: Crypto text = 464569
  - 464569^464307 % 698149 = 123 !!!
- Plaintext chosen by Bob would be the proposed secret session key K_{AB}
  - Only Alice has private key ***d*** and can decrypt the message to retrieve K_{AB}.
- Calculator: http://people.eku.edu/styere/Encrypt/RSAdemo.html

Signatures using RSA

Similar to encryption, but backwards.

- Use the same keys as before.
- Signing consists of modular exponentiation of plaintext ***m*** with ***d***
  - $s = m^d \% n$
- Verification consists of modular exponentiation of signature ***s*** with ***e***
  - $s^e \% n \rightarrow (m^d)^e \% n \rightarrow m^{ed} \% n \rightarrow m$
- As an example, Alice signs with Private Key ***d*** the value m = 123.
  - 123^464307 % 698149 = 91655
- Bob validates the signature using Alice's Public Key ***e***
  - 91655^3 % 698149 = 123

- EC math can be used with RSA
  - NSA has defined a "Suite B" that includes EC-RSA

PKC Issues & Problems

The need for certificates, CAs, chaining, revocation.

- Associating Public Keys with actual subjects.
  - When you encrypt, how do you know that you have the correct public key for Alice?
  - Are you sending a message securely to the wrong person?
  - We need a secure directory. DNS isn't good enough.
  - The X.500 directory service happened to be under development at the time.
- Only one public key is needed per person
  - You can have many names and many associations and many certificates
  - Protect the private key in hardware
- X.509 certificates securely associate X.500 names with public keys
  - A trusted Certificate Authority vouches for the association
- Certificate revocation is messy
  - CRLs, OCSP, …

PKC Applications

Encryption, Signatures, Authorization.

Examples of PKC usage:

- Public Key based Encryption:
  - Conditional Access in SA PowerKey. EMMs are encrypted with PKC.
  - SSL, ssh, IPSEC, etc. for connection encryption.
  - PGP for file encryption.
  - SecurID fobs do not use RSA, even though it is labeled as such.
- Signatures to establish identity.
  - STB firmware, cable modem firmware,
- Authorization certificates.
  - Difference between Identity and Authorization certificates (PACs)

Thank you for listening!!

- Questions?
- Send feedback to

  `rajaram@pejaver.com`

"Encryption software is expensive...so we just rearranged all the letters on your keyboard."