Cryptography, Authentication, and Integrity

CSC 348·648

WAKE FOREST
UNIVERSITY
Department of Computer Science

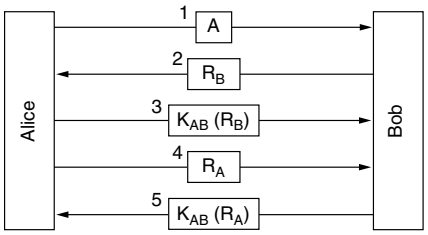
Spring 2013

Authentication

- We have discussed encryption to protect against passive attacks
  - Providing confidentiality by encrypting messages
- Can also provide authentication and integrity using encryption
  - Verify the **sender** and the **message**
  - Want to answer: *Is this who I think it is?* (authentication) and *Have the contents of the message been changed?* (integrity)

Shared Secret Key Authentication

- Assume Alice and Bob share a secret key k_{AB}
  - To establish the key they could use the telephone, etc...
- The protocol will use a **challenge-response** technique

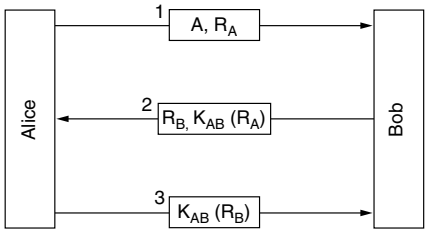


  - Alice first sends identity (A) to Bob
  - Bob responds with a challenge, a RN (r_B) in plantext
  - Alice returns the RN encrypted using k_{AB} (the response)

  - At this point Bob knows it's Alice, but Alice knows nothing

    What is one possible attack by Trudy?

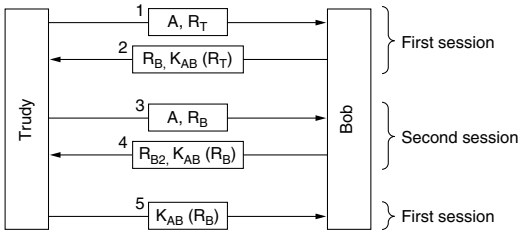
  - Alice selects a RN (r_A) and sends it to Bob
  - Bob returns r_A encrypted using k_{AB} (his response)
  - They can select a new key and establish a separate session
- The previous can be condensed to the following steps



Is the challenge-response protocol immune from attack?

Trudy's Revenge, Part 2

- Under certain circumstances, Trudy can apply a **reflection attack**
  - Assume Bob allows multiple simultaneous sessions
- Steps of the attack are



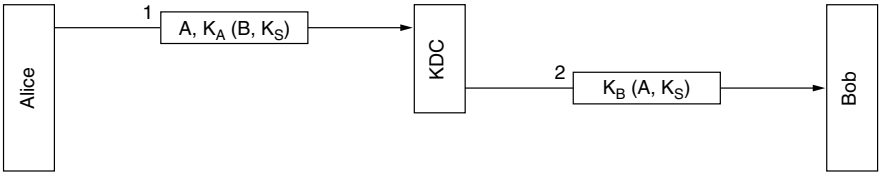
  - Trudy claims she is Alice and sends Bob A, r_T
  - Bob responds with his own challenge r_B
  - Trudy is *stuck*, she does not know k_{AB} for a response

  - Trudy opens a second session, supplying r_B as her challenge
  - Bob replies with the k_{AB} encrypted r_B
  - Trudy now has the response for the first session...
- The moral of the story is *designing a proper authentication protocol ain't easy*
- 3 general rules
  1. Have initiator prove identity before responder
     Was this the case in the first method?
  2. Have initiator and responder use different keys for proof
  3. Have initiator and responder draw challenges from different sets
     What?

Man-in-the-middle versus reflection, do both attack the same thing (privacy, integrity, or availability)?

Key Distribution Center

- Establishing a secret key with a stranger almost worked
  - However, such solutions typically do not *scale*
  - To talk to n people, you need n keys
- An alternative is a trusted **Key Distribution Center** (KDC)
  - Controls authentication and session key management
- A simple example, again assume Alice wants to talk to Bob



  - Alice selects a session key k_S and tells the KDC she wishes to talk to Bob

  - Alice encrypts the session key using another key (k_A) only she the KBC knows
  - KBC decrypts the message and sends Bob the session key encrypting it with another key (k_B) only the KBC and Bob knows

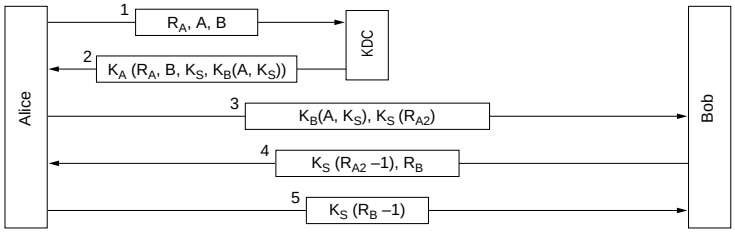
Has authentication occurred?

Trudy's Revenge, Part 3

- Unfortunately, the simple KDC system has a serious flaw
  - Assume Bob is Alice's banker, and Alice owes Trudy money
  - Alice establishes a secret key with Bob, then sends Bob an encrypted request for a money transfer to Trudy
  - Trudy copies the second message from the KDC system ($e_{k_B}(A, K_s)$) and the money request message that follows
  - Trudy then replays the two messages to Bob over and over, ...
- This is a **replay attack**
  - One solution is to include a timestamp (freshness);however, this requires clock synchronization
  - Another solution is a **nonce**, a unique one-time message number

Needham-Schroeder Authentication

- A multiway challenge response protocol
  - A KDC-based systems that includes nonces
- Authentication steps are as follows



1. Alice tells the KDC she wants to talk to Bob, message include a large random number (r_A) as a nonce

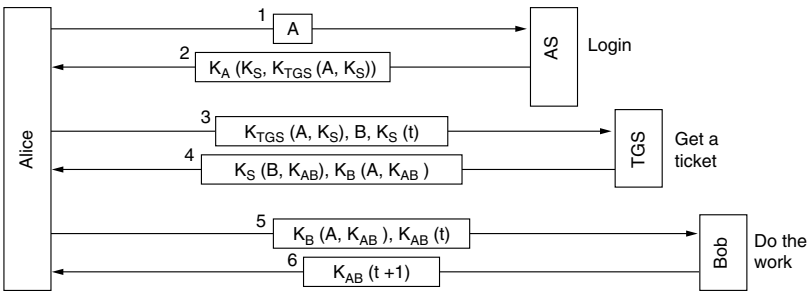
2. KDC sends Alice message with r_A, a session key, and *ticket*
   - r_A is for message freshness
   - Bob's identity is also included
   - The ticket is $k_B(A, k_s)$ (encrypted with Bob's key)
3. Alice sends the ticket and a new random number r_{A2} encrypted with the session key to Bob
4. Bob responds with $r_{A2} - 1$ and r_B encrypted with session key

   Why not return r_{A2}?
5. Alice knows this is Bob, she returns $r_B - 1$
6. Bob knows this is Alice

- Actually, this protocol is still susceptible...
  - Variations are used in commercial products

Kerberos

- Kerberos is a Needham-Schoerder variant
  - Designed by MIT to allow workstations to access resources
  - Named after multi-headed guard dog in Greek Mythology
- Involves two additional types of servers
  - Authentication Server (AS) - verifies user during login
  - Ticket-Granting Server (TGS) - issues *proof of identity tickets*



- Assume Alice sits down at a workstation
  1. Alice types in her login ID, which sends her name to the AS
  2. The AS responds with a session key and ticket ($k_{TGS}(A, k_S)$), encrypted using Alice's secret key
  3. The work station asks for Alice's password
     - Password is used to generate k_A
     - Using k_A session key and TGS ticket obtained
  4. Assume Alice wants to contact Bob
  5. Workstation sends TGS a request for a ticket with Bob
     - Message includes $k_{TGS}(A, k_s)$, proves Alice's ID
     - Includes a timestamp

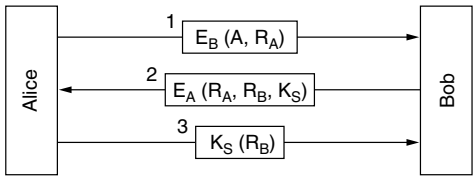
  6. TGS responds with session key k_{AB}
     - One is encrypted with k_S, the other with k_B

       Why encrypt the same thing twice?

  7. Alice can now establish a session with Bob
     - Message includes a timestamp

Authentication Using Public-Key

- Mutual authentication can be achieved using public-key
  - Assume Alice and Bob already know each other's public key
  - This is a **non-trival** assumption
  - They want to establish a session, then use secrete key (which is typically much faster than public key)
- Establishing a secret key would have the following steps
  - Alice encrypts her identity and RN (r_A) using Bob's public key



  - Bob decrypts and replies with r_A, r_B, and k_s encrypted with Alice's public key

  - Alice decrypts and verifies that r_A is correct

    What can Trudy do?

- The only true weakness of this approach is the public key
  - *How are public keys distributed in a safe fashion?*
  - This is one area of research, Public Key Infrastructure (PKI)

Authenticity

- Authenticity of many legal documents is determined by a signature
  - Notary public is used and photocopies do not count
  - For computer documents an alternative solution is needed
- Need a system where one party can send a *signed* message to another in such a way that
  - Receiver can verify the claimed identity of sender
  - Sender cannot repudiate the contents of the message
  - Receiver cannot have created the message
- This requirements are provided using **digital signatures**
  - Integrity - indicate whether a message has been altered
  - Authentication - indicate the person who signed

Private Key Authentication

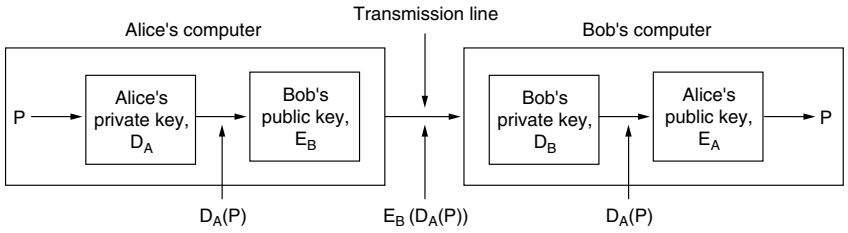
- It is possible to use secret key encryption for authentication
  - The key is known only to authorized people (*not trival*)
  - Only authorized people can encrypt and decrypt

    What about replay attack? How do you prevent?

    Can either party refute a message?

Public-Key Signatures

- Assume the public-key algorithm has the property
  - $d(e(p)) = p$ as well as $e(d(p)) = p$ (*RSA has this property*)
- If the above condition is true
  - Alice can *sign* and send the message $e_B(d_A(p))$, where Alice uses her private decryption key d_A and Bob's public key e_B
  - Bob receives the message and transforms using his private key, yielding $d_A(p)$, he then decrypts this using e_A



- *What happens if Alice denies sending p to Bob?*
  - Bob produces p and $d_A(p)$ in court
  - Judge can verify the message was encrypted by d_A by applying e_A to it

    What is Alice's final plea in court to get out of this?
- In principle, any public-key algorithm can be used for signatures
  - The de facto standard is RSA
  - In 1991 the NIST proposed using a variant of El Gamma for their **Digital Signature Standard**

Message Digests

- A complaint of signature methods is they couple disjoint functions
  - Authentication and secrecy
  - Often authentication is needed, but not secrecy
  - Encryption is often slow
- An alternative is the **one-way hash**
  - Takes an arbitrarily long p and generates fixed size message
  - No *key* is needed

    So where does the complexity lie?

  - There is **no** decryption (hence the one-way name)

    Without decryption how do you verify?

- Four objectives of one-way hash
  1. Given p, easy to compute $md(p)$
  2. Given $md(p)$, effectively impossible to find p
  3. Small change in p causes many bits to change in $md(p)$
  4. No two *reasonable* messages have same message digest
- Using a message digest with public key encryption
  - Alice would send $[p, d_A(md(p))]$
  - Bob would compute $md(p)$ and compare with received digest

Does MD provide integrity and/or authenticity?

Simple Hash Function

- All hash functions have a similar format
  - Input is seen as a series of n bit blocks
  - Input is processed one block at a time iteratively
  - Result is an n bit hash value
- Longitudinal redundancy check is one example
  - Bit-by-bit exclusive OR of every block

$$c_i = b_{i,1} \oplus b_{i,2} \oplus ... \oplus b_{i,m}$$

Where c_i is the i^{th} bit of the hash code, m is the number of n bit blocks in the input, and $b_{i,j}$ is the i^{th} bit of the j^{th} block

	bit 1	**bit 2**	...	**bit** n
block 1	$b_{1,1}$	$b_{2,1}$	...	$b_{n,1}$
block 2	$b_{1,2}$	$b_{2,2}$	...	$b_{n,2}$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$
block m	$b_{1,m}$	$b_{2,m}$	...	$b_{n,m}$
hash	c_1	c_2	...	c_n

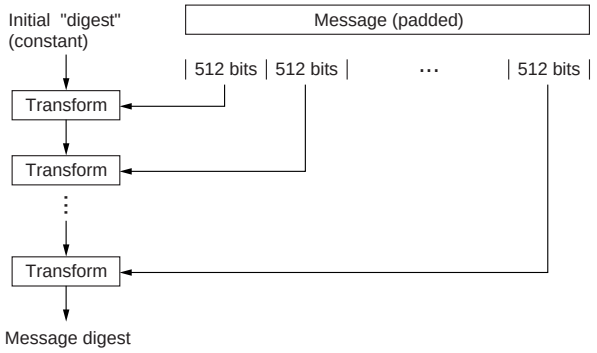
How many bits can change and still have a valid hash?

More suitable for error detection in transmission, why?

- Not a good one-way-hash
  - Single input bit change should change multiple hash bits

MDn

- There are a variety of message digest algorithms called MDn
  - Where n represents different methods, currently MD5
- MDn methods tend to have more in common with DES than RSA
  - Do not have a *formal* mathematical basis
  - Rely on complexity of the algorithm for strength
  - Every output bit is affected by every input bit
- The basic operation (MD4, MD5, and SHA)
  - Operates on 512 bits at a time (pad if necessary)
  - Digest calculation begins with initial constant
  - Value is combined with first 512 bits to produce a new value
  - Computation repeats until the final value created



- Main ingredient of MD5 is the transform
  - Inputs are the current 128 bit digest and 512 bits from message
  - Operates on 32 bit words so the current digest is (d_0, d_1, d_2, d_3)
  - The current message block is $(m_0, ..., m_{15})$

- Basic transform (compression module) can be divided into 4 passes
  - The first pass consists of 16 steps

$$
\begin{aligned}
d_0 &= (d_0 + f(d_1, d_2, d_3) + m_0 + t_1) \leftarrow 7 \\
d_3 &= (d_3 + f(d_0, d_1, d_2) + m_1 + t_2) \leftarrow 12 \\
d_2 &= (d_3 + f(d_3, d_0, d_1) + m_2 + t_3) \leftarrow 17 \\
d_1 &= (d_1 + f(d_2, d_3, d_0) + m_3 + t_4) \leftarrow 22 \\
d_0 &= (d_0 + f(d_1, d_2, d_3) + m_4 + t_5) \leftarrow 7 \\
d_1 &= (d_3 + f(d_0, d_1, d_2) + m_5 + t_6) \leftarrow 12 \\
&\vdots
\end{aligned}
$$

    where $f(\cdot)$ is a set of bitwise operations and t_i is a constant
  - Remaining passes have a similar form

Hash Security Summary

Below are publicly known attacks against popular hash methods. Rows in red are demonstrated attacks, while yellow rows are theoretical breaks.

- Collision attack (find two arbitrary inputs that will produce the same hash value)

Hash	Security Claim	Best Attack	Attack Date	Notes
MD5	2^{64}	$2^{24.1}$ time	7/2007	This attack takes seconds on a regular PC.
SHA-1	2^{80}	2^{51} time	2010	No successful reports of this attack yet.
SHA256	2^{128}	24 of 64 rounds ($2^{28.5}$)	11/25/2008	
SHA512	2^{256}	24 of 80 rounds ($2^{32.5}$)	11/25/2008	
MD2	2^{64}	$2^{63.3}$ time, 2^{52} memory	2009	Slightly less computationally expensive than a birthday attack, but for practical purposes, memory requirements make it more expensive.
MD4	2^{64}	3 operations	3/22/2007	Finding collisions almost as fast as verifying them.

- Chosen prefix collision attack (attacker can choose two arbitrarily different documents, and then append different calculated values that result in the whole documents having an equal hash value)

Hash	Security Claim	Best Attack	Attack Date	Notes
MD5	2^{64}	2^{39} time	6/16/2009	This attack takes hours on a regular PC.
SHA-1	2^{80}	2^{63} time	8/22/2006	Extends Wang's SHA-1 collision attack to partially chosen prefix collisions.
SHA256	2^{128}			
SHA512	2^{256}			

Algorithm	Output size (bits)	Internal state size	Block size	Length size	Word size	Rounds	Collision	Second preimage	Preimage
GOST	256	256	256	256	32	256	Yes (2^{105})	Yes (2^{192})	Yes (2^{192})
HAVAL	256/224/192/160/128	256	1,024	64	32	160/128/96	Yes	No	No
MD2	128	384	128	-	32	864	Yes ($2^{63.3}$)	No	Yes (2^{73})
MD4	128	128	512	64	32	48	Yes (3)	Yes (2^{64})	Yes ($2^{78.4}$)
MD5	128	128	512	64	32	64	Yes ($2^{20.96}$)	No	Yes ($2^{123.4}$)
PANAMA	256	8,736	256		32		Yes	No	No
RadioGatún	Up to 608/1,216	58 words	3 words	-	164	-	With flaws (2^{352} or 2^{704})	No	No
RIPEMD	128	128	512	64	32	48	Yes (2^{18})	No	No
RIPEMD-128/256	128/256	128/256	512	64	32	64	No	No	No
RIPEMD-160	160	160	512	64	32	80	Yes (2^{51}:48)	No	No
RIPEMD-320	320	320	512	64	32	80	No	No	No
SHA-0	160	160	512	64	32	80	Yes ($2^{33.6}$)	No	No
SHA-1	160	160	512	64	32	80	Yes (2^{51})	No	No
SHA-256/224	256/224	256	512	64	32	64	Yes ($2^{28.5}$:24)	No	Yes ($2^{248.4}$:42)
SHA-512/384	512/384	512	1,024	128	64	80	Yes ($2^{32.5}$:24)	No	Yes ($2^{494.6}$:42)
SHA-3	224/256/384/512	1600	?	?	64	24	No	No	No
Tiger(2)-192/160/128	192/160/128	192	512	64	64	24	Yes (2^{62}:19)	No	Yes ($2^{184.3}$)
WHIRLPOOL	512	512	512	256	8	10	Yes (2^{120}:4.5)	No	No

The Unix Encrypted Password System

- When you type in a password, Unix needs a way to verify
  - However, Unix **never** stores plaintext passwords
  - Instead, Unix stores the encrypted values in `/etc/passwd`
- Unix uses a secret key algorithm to compute a hash of a password
  - Unix never has to reverse the hash (encrypted password)
  - When you type in a password, it is hashed and compared
- Unix uses an algorithm (originally a DES-like) for encrypting
  - The password is converted to a secret key (first seven bits of the first 8 characters form the key)
  - Key is used with DES to encrypt the number 0
  - Add *salt* to hash and store in `/etc/passwd`

- Morris and Thompson added *salt* for variety
  - Salt is a 12 bit number that changes the DES encryption
  - When you change your password, a salt number is determined based on the time-of-day
  - Salt is converted into a two-character string and stored in the `/etc/passwd` file

    Why store the salt value?
- The result of adding salt is the same password can be encrypted 4096 different ways

  What is a disadvantage of having salt based on time-of-day?