Module III : Cryptography III : Email Security

1) Hash-functions (cryptographic primitive, properties, security)
2) Msg Authentication codes (authentication & integrity, private key)
3) Key Distribution : Private key setting
4) " " : Public " "
5) Email-security (PGP primitive)

Lecture 1 : Hash-functions

- can be applied to data of any length. (large i/p)
- Output is fixed length, usually very short. (o/p length does not matter)
- Relatively easy to compute $h(x)$, given x. (large i/p (1024, 2048 bits) → h(x) → 160/128 bits) — efficient
- function is deterministic. (same o/p if hash the same value twice) — compress (short)
- Infeasible to get x, given $h(x)$. One-wayness property
- Infeasible to find any pair x and y ($x \neq y$) such that $h(x) = h(y)$. collision resistance property. [forgery, signature will also collide]
  [large i/p, small o/p, many collisions might happen because of inherent property of hash-function] — hash & store hash passwords
- verify by matching the value of hash-value with the hashing of the file received. (hash is same)

* usually if a function is collision-resistant, it is automatically has one-wayness property.

Applications :

1) checksum for large amount of data [validate integrity of data]
2) Password hashing
3) Digital signatures
4) Msg authentication codes [authentication & integrity]
5) used also in RSA-OAEP & many other cryptographic constructions.

Hash output length

* (1) o/p should be of short length (2) o/p's should not collide with each other! so, what should be the minimal o/p length.
* How many people you have to sample to find 2 people having the same bday (bday paradox)

$\neq 365$, $\sqrt{365} = 23$ people (atleast 2 of them will have same bday)

* A collision can be found in roughly $\sqrt{N} = 2^{(n/2)}$ trials for an n-bit hash (with a probability $\geq 1/2$)

IMP * ∴ n should be atleast 160 bit.

* @ $\sqrt{2^{160}} = 2^{80}$ operations (extreme amount of operations) bday attack would be failed

Generic Hash-function - Merkle-Damgard construction

* It is not 100% sure that if h() is collision-resistant then H() will always be collision resistant. there has to be some sort of construction as the Merkle-Damgard construction. Here, H() is CR because h() is CR.

[even if o/p length is atleast 160 bit]

* Avalanche effect || → reason why H() is collision resistant.
* IV → fixed as hash is deterministic. (Public known constant)
* CV → chaining-vector

* CV_n → o/p of the hash-function. [160 bits here]
* large msg → (n X 512 bits hashed into 160 bits o/p) using this construct.
* Slide 6 illustrate the avalanche effect of the Merkel-Damgaard constructed hash-function. weak collision resistant.

Practical eg:

(Secure-hash algorithm)

(1) SHA-1	(2) MD5	(3) SHA-256 (256 long hash digest)
- o/p = 160 bits	- o/p = 128 bits	- o/p attack requires 2^{128} calls
- B day attack requires 2^{80} calls	- 2^{64} calls only (msg digest version 5)	* stronger version

* only difference in the Merkel-Damgaard construct is the compression function $ih(.)$ between these implementations.

Lecture 2: Msg Authentication codes (MAC)

* integrity & authentication (private-key cryptography)
* send (m, MAC); MAC is created ~~using~~ on m using the key shared b/w two-parties. [MAC + msg to the recipient, ~~specific~~ recipient verify MAC & to know used by intended sender + know that the msg has not been tampered with]
* Has to be deterministic to enable verification — unlike encryption scheme.
* MAC → small & secure as possible (efficiency) b/w deterministic
* Can not provide non-repudiation ? why ?
  ↳ Digital signatures (public model provide non-repudiation also)
* A can say that I did not create this MAC on msg using key, B has done it (recipient) as B also has same-key. No way to resolve dispute. Court cannot find who out of A & B created the MAC-value [deny sending what you sent]

(some key)

(signature using own private key (∴ you cannot deny))

MAC - functions

1) KeyGen - outputs a key (same key shared b/w A & B)
2) MAC - creates a checksum on m using key K.
3) verify - validates whether the checksum on m is computed correctly.

- Just create MAC & compare. (deterministic function)

Security - Notion

1) very similar to security notion for a digital signature scheme.
2) Existential forgery under (adaptively) chosen msg attack [query MAC from msg]

* prevent forgery ⇒ If attacker has many msgs & MAC pairs, he should not be able to create a new msg/MAC - pair.

MAC Based on Block cipher in the CBC mode - CBC Mac

* end result is used as the MAC - code.
* not an encryption scheme, dont need randomization property [∴ start with 0 ▷ should be fixed (IV) (not random)]
* E → Triple DES/AES

**

* IV is zero & only send result of last computation (Diff. from CBC)
* Secure against CMA (because of avalanche effect)

HMAC: MAC using HASH functions

* MAC derived from Hash.
* Double-hash
* Nested hash ↑ .. can be (SHA-256)

$$HMAC_K = H\left[(K^+ \oplus opad) \,||\, H\left[(K^+ \oplus ipad) \,||\, M\right]\right]$$

* other party repeats the same computation
* ipad & opad such that inner-key xor ≠ outer-key (even if key is same)
* Security → Nested-hash
* Single-hash → not secure
* Security related to collision resistance.
* CBC-MAC / HMAC

Lecture 3: key-Distribution

(Private-key setting)

- Private key → A & B share a secret key unknown to eaves.
- Public-key → A has a "trusted" (or authenticated) copy of Bob's public-key.

* • But, how does this happen in the first-place?

- A & B meet & exchange keys)
  - ↳ Not practical or possible (not physical close)
- we need key-distribution, first & foremost!!
  - ↳ Idea: make use of a trusted third-party (TTP)

Private-key

- Protocol assumes that A & B share a session-key K_A and K_B with a key-distribution centre (KDC)
- Alice calls Trent (trusted KDC) and requests a session-key to communicate with Bob.
  Trent generates random session key K and sends $E_{K_A}(K)$ to Alice and $E_{K_B}(K)$ to Bob
  Alice & Bob decrypt with K_A & K_B respectively to get K.
- key-distribution protocol.
- susceptible to replay-attack.

K_A & K_B ← Alice shares with trusted party; Bob shares with trusted party

[attacker can observe one session of communication b/w A & TTP & (B & TTP) & eavesdrop the encrypted keys & replay the keys in the next session & fool A & B into agreeing upon same keys in next session also

* If first session was compromised due to a malware or something somehow key was leaked out, by forcing A & B to agree upon the same keys in the second session, second session has automatically been compromised.
* Session keys are different. Compromise of one session should not come to the next session.

Needham-Schroeder Protocol

* random-nonce N_1 so that msgs are tied to the key.
* Reduces the impact of the replay attack.
* vulnerability → attacker can do a replay attack on the ticket $E_{K_B}(K \| ID_A)$ does not have nonce
* not fully susceptible to replay attack

&

Corrected version (with mutual authentication)

* Attack time-stamp in the ticket also.
* within certain-skew, then only msg is fresh
  time-stamp
* Now, secure !!
* Just TS should be synchronised with each other.

Lecture 4: Key Distribution (Public-key setting)

* Alice has a "trusted" (or authenticated) copy of Bob's public key. (How does Alice retrieve public key of Bob)
  - ↳ TTP (trusted third party)

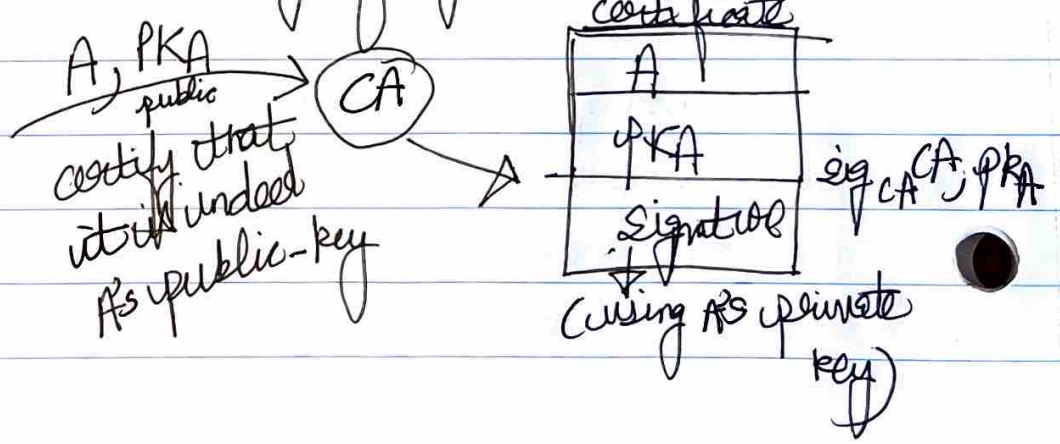
Public-key Distribution

- Public announcements (such as email)
  * — can be forged
- Public-Directory (website)
  * — can be tampered with
- Public-key certification authority (CA) (such as verisign)
  - (Third party) — CA issues certificates to the user.

(AUTHENTICATION OF PUBLIC KEY)

(malbory)

* M inserts his own public key instead of Bob's. M can intercept it & decrypt it. (using his private-key).
* ~~Eve~~ lies the identity of right owner.



* B can see the certificate, verify it's A's key.
* CA attests a bind b/w A & its identity
* DMV → license (name, identity, hair-color, DMV/CA signs the license)
  ↳ pub(verify license)

Naming & certificates

- Certification authority's vouch for the identity of an entity – Distinguished names (DN) → unique-name
  /O = UAB/OU = CS/CN = Nitish Sadana
  – Although CN may be same, DN is different.
  (common-name)

Types of certificates

- CAs vouch at some level the identity of the principal.
- Eg – Verisign
  – Class 1 – Email address
  – Class 2 – Name and address verified through database.
  – Class 3 – Background check
    ↳ periodically (eg → every night)

* Certificate Revocation → certificate revocation list (CRL)
  list of every certificate that has been revoked but not expired.
  → other mechanisms to validate validity of certificate if it has been revoked or not
* OCSP (online certificate status protocol)
  (real-time) (query certs)

* X.509 Certi (standard)

Advantages of CA over KDC

(only when e.g. requesting an certificate / not for communication)

* CA does not need to be online all the time.
- CA can be very simple computing device.
- If CA crashes, life goes on (except CRL)
- Certificates can be stored in an insecure manner.
* Compromised CA cannot decrypt msgs [does not know private key, only knows public-key, so compromised communication ✗]
- Scales well.

* KDC [has access to private-key, if compromised, all communication gone!!] compromised

Public-key Infrastructure (PKI)

* Digital certificates + public-key cryptography + certificate-authority

* Google's server certificate
* Browser verifies validity of Google's certificate before we do communication with it. (Gmail-server)

Lecture 5: Email Security via PGP

(I) Vulnerable:
1) eavesdropping ; 2) Impersonation / spoof (create email on behave of other people)

(II) Properties
1) Confidentiality — protection from disclosure [Hide email communication from attacker]
2) Authentication — of sender of msg
3) Integrity — protection from modification
4) Non-repudiation of origin — protection from denial by sender.
   ↳ ✓ for Digital signatures [Non-repudiation, authentication & integrity]
   ✗ MAC [only integrity & authentication, not non-repudiation]
   + encryption (for confidentiality)

(III) Pretty good privacy (PGP)
1) open-source, freely-available software package for secure email
2) De-facto standard for secure email
3) Developed by Phil Zimmermann
4) Selected best available crypto algo to use
5) Runs on a variety of platforms like Unix, PC, Mac & other systems
6) originally free (now also have commercial versions available)

(IV) PGP Operation – [Authentication + Integrity + non-repudiation]

Just use digital signatures:

1) Sender creates msg.
2) Generates a digital signature for the msg. (using private key)
3) Use SHA-1 to generate 160 bit hash for the msg.
4) signed hash with RSA using senders private key and is attached to msg.
5) Receiver uses RSA with senders public key to decrypt and recover hash-code
6) Receiver verifies received msg using hash of it & compares with decrypted hash-code.

Signature = RSA [hash(msg)] (private key) + msg

(V) PGP operation – Confidentiality (only)

private key session

1) Sender generates a msg.
2) Generates a 128-bit random no. as session-key.
3) Encrypts the msg using CAST-128/IDEA/3-DES in CBC mode with session-key
4) session key encrypted using RSA with recipients public-key and attached to the msg.
5) Receiver uses RSA with private-key to decrypt & receive session-key.
6) Session-key is used to decrypt the msg.

* encrypting using public-key [RSA-OAEP] quite intensive if the msg is very long. ∴ we don't do it. computation power ↑↑

* session-key (short) as it is private an efficiently encrypt long-msgs instead of using public-key (quite long) to encrypt msgs. But, use the public-key to encrypt the session-key [& as session-key is short, computation remains efficient only].

$$\left[Enc_K(m),\ Enc_{PK_B}(K)\right] \xrightarrow{\text{to Bob}} (K, m)$$

(VI) PGP operation – confidentiality & authentication

- can use both services on same msg.

1) create signature & attach it to the msg.
2) Encrypt both msg & signature
3) attach RSA encrypted session-key

sequence is preferred because–

1) one can store the P.T msg/file and its signature
2) no need to store the C.T for future signature verification

$$\underbrace{Enc_K(m, S)}_{S \text{ on msg using private key}},\ \underbrace{Enc_{PK_B}(K)}$$

* could have first encrypted the msg & then signed but the previous approach is preferred beco.
* (Sign & then encrypt)
* Can store signature/msg pair for future verification [In case something goes wrong, later verify it!]

(VII)

PGP operation – Compression

1) PGP compresses msgs to save space for email txn & storage.
2) By default, PGP compresses msg after signing but before encryption.
   - So can store uncompressed msg & signature for later verification
* - Encryption after compression strengthens security (beco compression has less redundancy)
3) uses ZIP compression algo.

* $$\left[Enc_K\left(Z\left(m, s\right)\right), Enc_{pk_B}(K) \right]$$
* verify signature if needed at later point of time

(VIII)

PGP operation - Email Compatibility

1) When using PGP will have binary data (8-bit octets) to send (encrypted msg) etc.
2) However, email was designed only for text.
3) Hence, PGP must encode raw binary data into printable ASCII characters.

4) uses radix-64 algo
   ↳ maps 3 bytes to 4 printable chars
   ↳ also appends a CRC (verify integrity of the msg)

5) PGP also segments msgs if too big (max length 50,000 octets)

encoding → decoding
(binary → text) (text → binary → cryptography, all properties like authenticity etc. are verified)

* store msg & signature at receiver for future verification at the receive-side.

(IX) PGP session-keys

1) Need a session key for each msg
   - of varying sizes: 56-bit DES, 128-bit CAST or IDEA, 168-bit for triple-DES

2) Uses random i/ps taken from [user-actions]
   - actual key hits.
   - keystroke timing of user
   - mouse movement.

(really true random-values → to create session-key (rather than pseudo random values))

(X) PGP key Distribution

1) Public key for encrypting session key / verifying signature

2) Where do these key come from and on what basis they can be trusted?

Signature, Compress, Encrypt

(XII) PGP key Distribution:

1) PGP adopts a trust model called the web of trust.
2) No centralised authority. (De-centralised)
3) Individual signs one another's public keys, these "certificates" are stored along with keys.
4) PGP computes a trust-level for each public-key in key-ring. [scale]
5) User interpret trust level for themselves.

* More scalable, certifying public-keys
  ↳ no authority.

(XIII) Issues:

1) original intention was email users would contribute to web of trust. (Not many people use PGP encryption)
2) Reality is that web is sparsely populated. [web keys ← Not fully vouched-for]
3) How should security-unaware users assign & interpret trust-levels?
4) Latest version of PGP supports X.509 certs.

(C.A)

* setting public-key & sending encrypted msgs via email to T.A.