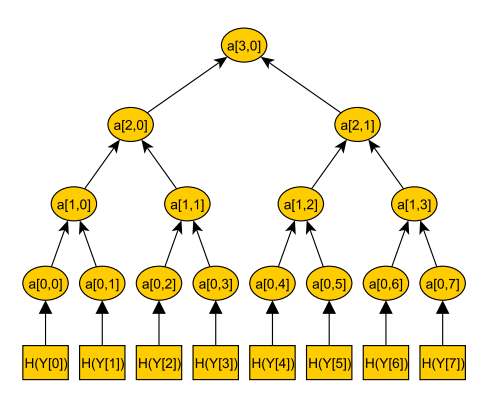
**1) Merkle Hash Tree**

•**Construct a Merkle hash tree for 8 messages m1, m2, ..., m8. (Draw the figure).**



(Source: https://www.emsec.rub.de/media/crypto/attachments/files/2011/04/becker\_1.pdf)

**KeyGen:** Generating both public key (Xi) and private key (Yi) for 8 one-time signatures.  
 1) Generate private keys Xi using One-Time signatures scheme KeyGeneration e.g. Lamport or Winternitz scheme.   
 2) Hash each of the private keys to get a one-time public keys Yi .

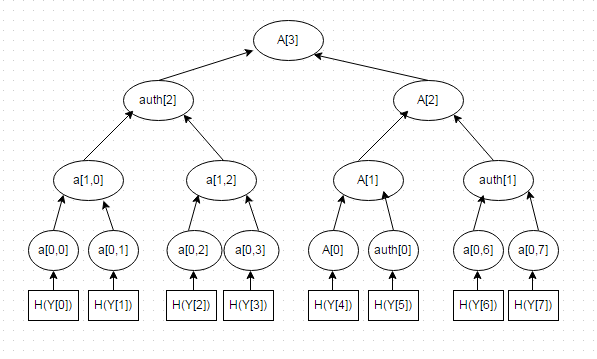
**Tree Construction:** We have acquired both public and private keys. We can now construct the Merkel hash tree  
 3) For each public key Yi, a hash value is compute hi  = H(Yi).  
 4) Each hi  corresponding to the lowest node with in a hash tree. a0,i  = hi  
 5) To compute an upper node, it can be computed by concatenating its children nodes and hash the value.   
 For an example: a1,0  = H(a0,0 || a0,1) then for an upper node, a2,0  = H(a1,0 || a1,1) so on...

**Signature Generation:**   
 6) To sign a message mi, we use a pair of public and private keys (Xi , Yi).  
 Note: The leaf of the hash tree corresponding to a public key Yi is a0,i = H(Yi)   
 7) There is a path from a0,i to the root, let's call it path A0.  
 (There are total of A0, A1, ..., An paths).

8) To compute the root path A, we need to know every child of the nodes A0, A1, ..., An.   
 (We know that Ai is a child of Ai+1 and to calculate Ai+1 of the path A we need to know both of the child nodes of Ai  and Ai's siblings) Thus, Ai+1  = H(Ai || *authi* )   
 Note: the sibling node of Ai is also known as *authi .*    
 9) All of the *auth* node and a one-time signature *sig* is the signature  
 signature = (*sig* || *auth2* || *auth3* ||... || *authn-1*)

• **Suppose a sender S uses this Merkle hash tree to authenticate these messages to a receiver R. What should be done before this tree can be used?**

Assuming a receiver knows the one-time public key, the message M, and the signature sig.   
**Signature Verification:**  
 1) Receiver verifies the one-time signature, sig, of message M.  
 2) If the sig is valid, then the receiver will compute the hash value of the public key. A0 = H(Yi)  
 3) The receiver will then continue to construct the a hash tree by concatenating with the *authi* and hash the result. At the end, the receiver should obtain a public key if the signature is valid.

**• How would one authenticate message 4?**   
(See general steps on the answer above)  


• **Merkle hash trees have an additional level of hash in the leaves. Is this necessary? Why?**

It is necessary because it introduces diffusion to scheme to prevent bit manipulation from adversary that may occurs.

https://www.emsec.rub.de/media/crypto/attachments/files/2011/04/becker\_1.pdf

**2)**

In general Schnorr Signature works as the follwing: **Signing:** 1) Choose a random number k   
 2) Compute r = gk ; g is a generator  
 3) Encrypt e = H(M||r)  
 4) Sign s = (k - xe)  
The signature is (s, e) or ( k-xe, H(M||r)) **Verifying:** 1) rv = gsye  
 2) ev = H(M||r)  
Check if ev = e

But for our case, Schnorr signature will look something more like this:  
**Signing:**  
 1) Choose a random number k   
 2) Compute r = gk (where r can be reset)  
 - potentially causes collision with the previously generated number  
 3) Encrypt e = H(M||ts||r)  
 - the timestamp is a known value and is hashed along with the message.  
 - it provides adversary with random oracle queries (one query for each signature)  
 4) Sign s = (k -xe)  
 which is equivalent of s = (k - x\*H(M||ts||r) )  
 5) A signature is (s,e) or ( k-x H(M||ts||r), H(M||ts||r))  
  
After observe a sufficient amount of signatures, an adversary knows what H(M||ts||r) looks like in each packet; he can solve for the value of k via s = k-x H(M||ts||r). (Assuming an adversary happens to be able to find two signature where x happens to be the same) Now what an adversary has to do is obtaining any insight regarding the hash function. A timestamp is a known value and is hashed along with the message and random number r. Given an adversary has an access to random oracle, he can guess what r and M might be via statistical analysis. On the algebraic side, the paragraph above be deduced to the following system of equations:

s0 = ak-1 + ϒk-1 mod q ; ϒ = xr, a = H(f(k)||ts)  
 s1 = bk-1 + ϒk-1 mod q ; b = H(f(k)||ts)

**Note:** ϒ = xr because an adversary can find a signature collision due to the bad PRGN.  
 a, b are the hash value which depends on k (which is a bad PRGN) and a timestamp which is a   
 known value.  
With two equations and two unknown, a and ϒ, an adversary can obtain the k by solving the system of equations.

**3) Basic security models for authentication primitives**

•**Explain EU-CMA experiment for a MAC scheme.**A security scheme that is EU-CMA is known to prevent an adversary from 2 case scenario:  
 1) A security scheme should prevent adversary with a knowledge of the key to generate messages and acquire their valid signature. (Note: the adversary does not need to have know ledge of the key)  
 2) A security scheme should prevent an adversary from generating some messages at his own with a valid signature. (Existential forgery)

**EU-CMA experiment (MAC):** 1) A signer generates a random n-bit key and shares it with an adversary: k <- {0, 1}n2) An adversary receives a message M  
 3) An adversary queries to authenticate a message M: authk(M) -> (M,t)  
 4) An adversary can repeats step (2) and (3) if needed  
 5) An adversary guesses a new pair of (M', t') and sends (M', t') to signer  
 6) An adversary wins if he successfully generate M', t' such that authk(M') is accepted

•**Explain EU-CMA experiment for a digital signature scheme.  
EU-CMA experiment (Digital Sign):** An experiment to test if a signature scheme is EU-CMA secure.   
Assuming it is computationally infeasible for an adversary to compute all signing keys and find a collision of a key that yields a passing signature verification.  
 1) A signer generates n-bit signing key Sk and verification key Vk : Gen(1n) -> (Sk,Vk)2) An adversary receives verification key Vk 3) An adversary receives a message M  
 4) An adversary signs M with Sk  : s <- Sign(Sk , M)  
 5) An adversary can repeats steps (3) and (4) if needed  
 6) An adversary sends a signed message to a signer: (M', s')  
 7) An adversary wins if a signer verifies that (M', s') is true and M' is not signed by the signer:  
 Ver(Vk, M', s') = true; M' != M.

**Note:**  The scheme is EU-CMA secure if the probability of adversary winning is negligible.

• **Why adaptability is important, and what are non-triviality and validity conditions?**

**Adaptability** in the context of security scheme experiment means that a challenge is allowed to choose the next message after seeing the previous ciphertext. It is important because it allows an adversary/challenger to be able to carefully craft message in a way that it would give him on how the security scheme works.

**Non-Trivial and validity condition** means that adversary does not get away with (verification = true) by using the message that the signer previously signed.

**4) Do you see any synergy between Elgamal encryption scheme and DH key exchange protocol?** •**Explain how one implicitly uses another to achieve its objective.**

**Diffie Hellman Key Exchange:**

Alice Bob  
 a <- Zq\* b <- Zq\*  
 A <- αa mod p B <- αb mod p

Exchange A ---->  
<---- Exchange B

KAB = αab KAB = αab

**ElGamal:**Let's assume Bob wants to send an encrypted message to Alice **KeyGen:** Generating a public and private key for ElGamal scheme1) Pick a number from a prime set as a private key: a <- Zq\*2) Compute an intermediate exchange key A: A <- αa mod p 3) Sk <- a; Pk <- (A, α, p)  
**Notice:** Alice key generation in ElGamal is exactly the same as how Alice generates her private and public keys in DH.

**Encryption:** Encrypting an information X  
Recall: Bob knows A, α, and p (they are contained in the public key)  
 1) Pick a number from a prime set as a personal private key: b <- Zq\*  
 2) Computer an intermediate exchange value: B <- αb mod p  
 3) Encrypt a message by multiplying with a given Ab: Y <- XAb mod p  
 4) Concatenate B and Y, ready to send: C <- (B, Y)  
**Notice:** step (1) and (2) is the same as how Bob would generate his keys  
 step (3) is different from original DH key exchange but if you examine closely, you will see that the Encrypted message Y is merely X(αa)b = X(αab) which is a shared key in DH key exchange multiplied with a piece of information.  
 step (4) is to send the message from Bob to Alice with an extra intermediate value B to allow Alice to decrypt the message.

**Decryption:** Alice decrypting message from Bob  
Recall: Alice knows B and Y from Bob  
 1) Compute X <- Y(Ba)-1

Note: Alice knows that Y = X(αab). All she has to do is to divide the ciphertext with a shared key.

**5)Cryptographic hash chains and their usage in schemes/protocols**

**• In class, we discussed a simple method for One-Time Signatures (OTSs), in which hashchains can be used to reduce key storage from O(L) to O(1). Explain how it works.**

In class, we discussed about the Lamport signature scheme and how can we optimize the size of the key storage from O(L) to O(1). The improved version of Lamport signature is known as Winternitz One-Time Signature scheme. The scheme allows you to trade key storage size for time complexity.

**KeyGen:** 1) Parameter w is chosen from a set of a positive integer  
 2) Calculate parameter t where   
 t = s/w + (log2(s/w) + 1 + w)/w (round up)  
 3) Choose t random number X1, ... , Xt with the length of s-bit  
 4) Concatenate all of the value of X1, ... , Xt to obtain a private key X  
 5) Generate a public key Yi from Yi = H2w-1(Xi) for i = 1,..., t  
 6) Concatenate all the value of Y1, ... , Yt to obtain a one-time public key Y

https://www.emsec.rub.de/media/crypto/attachments/files/2010/04/da\_bensmann.pdf  
https://www.emsec.rub.de/media/crypto/attachments/files/2011/04/becker\_1.pdf  
http://en.wikipedia.org/wiki/Lamport\_signature#Verifying\_a\_signature

**• Forward-security is a desirable property for an authentication scheme, and hash-chains can be used to achieve forward-secure, as well as other desirable properties such as aggregation. A good example for such an approach is given in the below with FssAgg-MAC scheme, which is briefly discussed during the class. Please describe this scheme in detail, how it achieves forward-security, O(1) tag size (i.e., aggregation) and why it is efficient compared to its PKC-based counterpart.**

**FssAgg overview:**  
The FssAgg is a forward-secure cryptographic scheme that allows the data to immediately be signed and sent in a packet of a constant size with the minimal overhead. The FssAgg based on private/public key pair structure with the keys value pre-computed and stored ready to be used; it can be used to authenticate multiple message when the public verification is not required.

**FssAgg MAC:** 1) FssAgg.Kg: the scheme uses a symmetric key generation to produce its signing and verifying key (both of them are the same): Sk = Vk 2) FssAgg.Asig: message M is arrived and is signed using a one-way hash function to   
 generate a σ0,i . To keep on signing, the scheme then fold σ0,1 , σ0,2 into σ1,i   
 3) The scheme updates its key value using the latest output of hash as a random number seed

**Note:** FssAgg is more efficient than PKC because the signing and verifying key generation is based on symmetric key generation. Thus, there is less overhead to keep track which allows the FssAgg to generate the key as it goes.

**6) Kerberos:**

**• When Bob receives a ticket from Alice, how does he know it is genuine?**Bob can be sure that the ticket he receives from Alice is genuine by checking if ticket is authenticated by KDC. In fact, KDC is the one who generates a ticket for Alice to send to Bob. Thus, Bob can check the ticket's genuinity by decrypting the ticket using the key he shared with the KDC.

**• When Bob receives a ticket from Alice, how does he know it came from Alice?**   
Once Bob receives a ticket and he knows exactly where the ticket is from because the KDC generates a ticket that has the following fields: {Alice's ID/name, Bob's service name/ ID , Alice's IP address, Timestamp, Lifetime of ticket, Bob's service session key}. Bob can authenticate that Alice is Alice after decrypt the package and start communicating with alice.

**• When Alice receives a replay, how does he know it came from Bob (that it is not a replay of an earlier message from Bob)? • What does the Ticket contain that allows Alice and Bob to communicate securely?**Alice will make sure that the packet she receives does not match with any previously stored ticket form Bob. Moreover each ticket contains a timestamp and the period of validation for a ticket which prevents a ticket to be reused.

http://www.roguelynn.com/words/explain-like-im-5-kerberos/

**7) Needham-Schroeder (Extended version in slides), Otway-Rees and Kerberos**

• **In the Needham-Schroeder protocol, how is Alice authenticated by the KDC? How is Bob authenticated by the KDC? How is the KDC authenticated to Alice? How is the KDC authenticated to Bob? How Alice is authenticated to Bob? How Bob is authenticated to Alice?**

The KDC authenticate Alice by have Alice send a randomly generated number to KDC. KDC is then generates a ticket to Alice that contains Bob's ticket inside along with random number N. The ticket contains the following fields: {Alice's random number, Alice's ID/name, Bob's service name/ ID , Alice's IP address, Bob's service session key}.  
**Note:**  The ticket fields are almost identical to Kerberos' ticket fields except there is a random number in place of a timestamp and a ticket validation period.

After Alice receives the ticket from KDC, she decrypts the ticket with her private key, obtains a session key with KDC, forward an encrypted Bob's ticket directly to Bob. Bob then decrypts the ticket using his private key to obtain the session key and Alice's random number N. Bob then computes N-1, generate another random number M, concatenate (N-1 || M), encrypted it, and send the packet to Alice. Alice is then decrypted the (N-1 || M) with a session key, computes M-1, encrypts it, send it to Bob. Once bob receives the (M-1), he can verify that Alice is real.

• **Why in the Needham-Schroeder protocol, Alice is the party that is in contact with the KDC, but in the Otway-Rees protocol, Bob is the party that is in contact with the KDC?**

It is because in expanded Needham-Schroeder protocol Bob is the one who authenticate himself to Alice first which is prone to a reflection attack. To solve the problem, Otway-Rees protocol has Alice authenticate herself to Bob first. Then, Bob will be the one who initiate communication with KDC regarding Alice. This way Alice can be sure that Bob is Bob and Bob can be sure that Alice is Alice.

• **Why do we need only one time-stamp in Kerberos instead of four nonces as in the Needham-Schroeder protocol or three nonces as in Otway-Rees?**

Technically you need both a timestamp and a validation period of a ticket. With the two components in Kerberos protocol, both Alice and Bob can use the information to prevent a replay attack.

**8) PKI Concepts:**

**• What are the core components of a PKI? Briefly describe each component.**   
PKI or Public Key Infrastructure is the entire system of Certificate Authentication with all of its required support mechanisms. The core components of a PKI are following:  
 1)Digital Certificates: A digital identify that is trusted and known by CA.  
 2)Public and Private Keys: A simply public and private keys. The public key is contained in a certificate while a private key is kept secret and is used to sign a message.   
 3)Secure Sockets Layer: A standard internet protocol.   
 4) Certificate Authority (CA): A third party that acts like a trusted, independent who provides/revokes digital certificates

https://networklore.com/components-of-pki/  
http://docs.oracle.com/cd/B12037\_01/network.101/b10777/pki.htm#1006146

**• Discuss the trustworthiness of root certificates provided by browsers.**

They are more or less trustworthy given they have to follow some Extended Validation baselines that is required by CA/Browser Forum in order to become a CA. Though, The first CA/Brower complete baseline requirements was first enforced in 2012 with issuance and management of publicly SSL certificates. Since then, there has been annually evaluation of the Baseline Requirements under the observation of AICPA/CICA.

https://casecurity.org/ca-browser-forum-standards/  
http://www.zdnet.com/article/chrome-does-certificate-revocation-better/  
http://en.wikipedia.org/wiki/Certificate\_authority#CA\_compromise

**• What is the purpose of the X.509 standard and what is a certificate chain? How is an X.509 certificate revoked?**  
**Purpose:**  
The purpose of the X.509 is to give some insights into practical aspects of PKI. PKI roughly consists of: 1) Certificate Algorithm: this section specifies which signing algorithm is used. (RSA, SHA-1,..)  
 2) Issuer: Who issues the certificates.  
 3) Period of Validity: How long is the key is valid for.  
 4) Subject: To whom is this message.  
 5) Subject's Public Key  
 6) Signature: The signature over all other fields

**Certificate chain:**   
Ideally, there should be one CA who issues certificates for all of the users. Though, that is not possible in the real world practice; there are many different CAs and each CA can have different format of the certificate. That is when the certificate chain comes into play.

Let's say Alice and Bob are on a different CA and they want to communicate.

|  |  |
| --- | --- |
| Alice | Bob |
| Pk,CA1 | Pk,CA1 |
|  | CertB = [(Pk,B,IDB),sigkpr,CA2(Pk,B,IDB)] |
| <------CertB | |

**Note:** Alice cannot authenticate Bob because she only has her public key from CA1 and Bob has the public key of CA2

What Alice has to do now is she requests for CA2 public key and within it contains a signed certificate from CA1.

|  |  |
| --- | --- |
| Alice | CA2 |
| RQST(CertCA2)----> | |
| <----CertCA2 | |
| VerPk,CA1(CertCA2)  VerPk,CA2(CertB) |  |

At the end, CertCA2 looks like CertCA2 = [(Pk,CA2 , IDCA2), SigPk,CA1(Pk,CA2 , IDCA2)]  
Now, Alice can verifies Bob's key.

**Certificate Revocation:**   
In general, the KDC would generate a list of certificate that is no longer valid (called Certificate Revocation List, or CRL), and distribute the list to the users. Ideally, it would be the most secure if KDC distribute CRL to a user every time the users wants to communicate but that would require KDC to be involve in every communication. Alternatively, KDC sends out CRL periodically after a fix amount of time e.g. daily, weekly, etc.

**RSA digital signatures and their properties: Aggregate signatures enable O(L) signatures to be compressed into a single, compact (i.e., constant-size) verifiable signature.**

**• In RSA, it is easy to aggregate signatures computed under the same private key. What is the name of this scheme, and how this aggregation is performed?**

The name of the scheme is PISB-CSA-RSA. The following is how the PISB-CSA-RSA works:  
**KeyGen:** Both data owner and the sever have to generate their own private and public keys.1) Randomly pick two large prime numbers: p, q <- Zq\*  
 2) Computes the value of n and φ where n = p∙q and φ = (p-1)(q-1)  
 3) Pick a random number e: e <- Zn\* such that 1 < e < φ and GCD(e, φ) =1  
 4) Pick a decryption key d: d <- e-1 mod n  
 5) The public and private keys are Pk <- (n,e); Sk <- (n,d)  
**Note:** PISB-CSA-RSA requires two sets of public-private keys. One for server, another for data owner. The steps (1) - (5) has to be executed twice by two different entities. With the restriction that the value n of the server has to be larger than the value n of the data owner  
 6) The system public and private keys are Pk <- (PkOwner, PkServer), Sk <- (SkOwner, SkServer)

**Singing (Data owner side): Init(M,Sk)** The data owner has acquired Sk, n,and a message M 1) Compute Sj <- [H(Mj)]d mod n for j = 1,..., l; l = length of the message  
 2) The data owner sign a message V <- (M, S) and send it to the server  
**Singing (Server owner side): Sign(SkServer, M, V)** The sever received V and a message Mσ 1) The server then sign the message using the information from V and M: σ <- ∏j=1l sj mod n.  
 2) The server then compute h <- H(M||PkServer)  
 3) y <- (h + σ) mod nServer   
 4) γ <- yd mod nServer   
**Note:** The signing algorithm is different from regular RSA signing algorithm that PISB-CSA-separate the regular RSA algorithm into two parts: the data owner and the server. The data owner hash the message, encrypted it with his private key then send the data for the server to sign. The server then sign the message, hash it, and encrypts it.

**Verifying:** The receiver knows γ  
 1) Decrypt the message using RSA decryption: γe mod nServer  
 2) Compute the signature: σ <- (y ' - h') mod nserver   
 3) Check if (1) and (2) yield the same result

**Note:** This is moderately the same as regular RSA signature verification.

**• In cloud computing environment several tuples belonging to different users are stored together. Therefore, it is desirable to aggregate signatures computed with different keys belonging different users. Is it possible to use the RSA aggregation strategy discussed above for this purpose? If it is possible, show how it can be done. If it is not possible, explain the reason in detail.**

It seems like the strategy is developed specifically to be used in the case as stated. For an example, a mobile device has less computation power than a server. Thus, it can do a simple encryption (such as hash) relatively quickly then the data is sent to a server to do the signing and double encrypt the result to make sure that the data is secured.