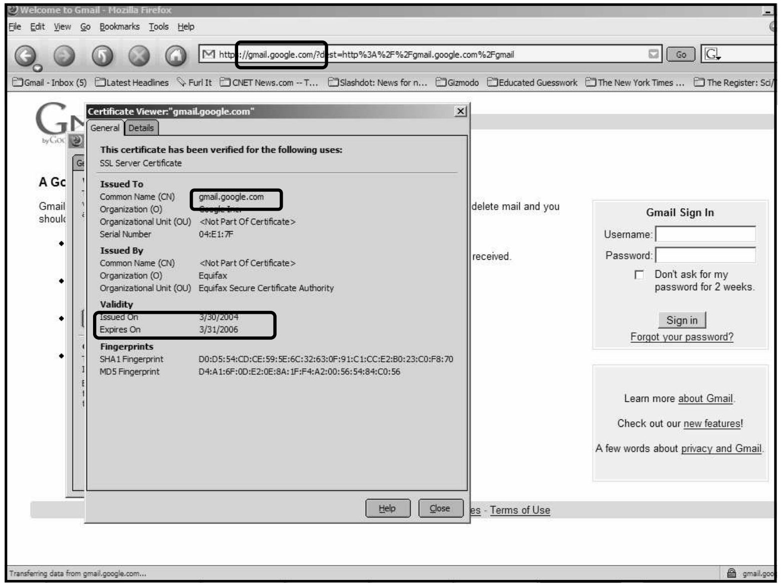
Introduction to Cryptography
Lecture 10

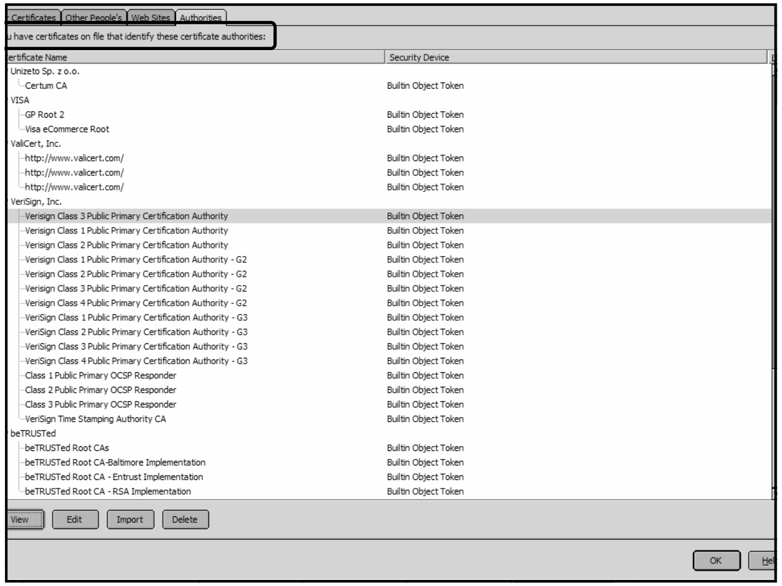
Public Key Infrastructure (PKI), hash chains, hash trees. Primality testing.

Benny Pinkas

Certification Authorities (CA)

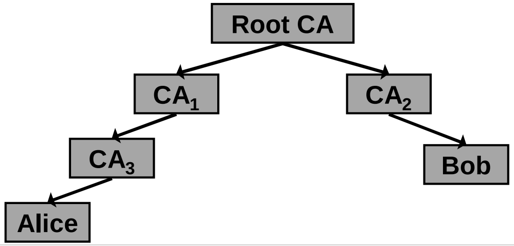
- How can users verify that a public key PK_v corresponds to user *v*?
- A Certificate Authority (CA) is trusted party.
- All users have a copy of the public key of the CA
- The CA signs Alice's digital certificate. A simplified certificate is of the form *(Alice, Alice's public key)*.
- The CA can work offline.
- When a user wants to communicate with Alice, it must obtain her certificate. Either directly from her, frm the CA, or from a public repository.





Public Key Infrastructure (PKI)

- Monopoly: a single CA vouches for all public keys
- Monopoly + delegated CAs:
  - top level CA can issue certificates for other CAs
  - Certificates of the form
    - [(Alice, $PK_A)_{CA3}$, (CA3, $PK_{CA3})_{CA1}$, (CA1, $PK_{CA1})_{TOP\text{-}CA}$]



Revocation

- Revocation is a key component of PKI
  - Each certificate has an expiry date
  - But certificates might get stolen, employees might leave companies, etc.
  - Certificates might therefore need to be revoked before their expiry date
  - New problem: before using a certificate we must verify that it has not been revoked
    - Often the most costly aspect of running a large scale public key infrastructure (PKI)

Certificate Revocation Lists (CRLs)

- A revocation agency (RA) issues a list of revoked certificates (i.e., “bad” certificates)
  - The list is updated and published regularly (e.g. daily)
  - Before trusting a certificate, users must consult the most recent CRL in addition to checking the expiry date.
- Advantages: simple.
- Drawbacks:
  - Scalability. CRLs can be huge. There is no short proof that a certificate is valid.
  - There is a vulnerability windows between a compromise of certificate and the next publication of a CRL.
  - Need a reliable way of distributing CRLs.
- Improving scalability using “delta CRLs”: a CRL that only lists certificates which were revoked since the issuance of a specific, previously issued CRL.

Explicit revocation: OCSP

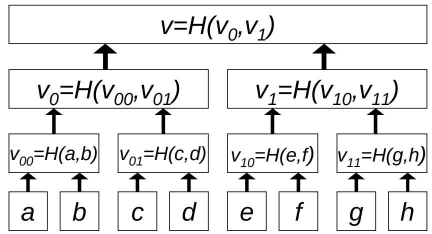
- OCSP (Online Certificate Status Protocol)
  - RFC 2560, June 1999.
- OCSP can be used in place, or in addition, to CRLs
- Clients send a request for certificate status information.
  - An OCSP server sends back a response of "current", "expired," or "unknown“.
  - The response is signed (by the CA, or a Trusted Responder, or an Authorized Responder certified by the CA).
- Provides instantaneous status of certificates
  - Overcomes the chief limitation of CRL: the fact that updates must be frequently downloaded to keep the list current

Certificate Revocation System (CRS)

- Certificate Revocation System (Micali'96)
- Uses a hash chain
  - The certificate includes $Y_{365} = f^{365}(Y_0)$. f is one-way.
  - On day d,
    - If the certificate is valid, then $Y_{365-d} = f^{365-d}(Y_0)$ is sent by the CA to the certificate holder or to a directory.
    - The certificate receiver uses the daily value ($f^{365-d}(Y_0)$) to verify that the certificate is still valid. (how?)
- Advantage: A short, individual, proof per certificate.
- Disadvantage: Daily overhead, even when a cert is valid.

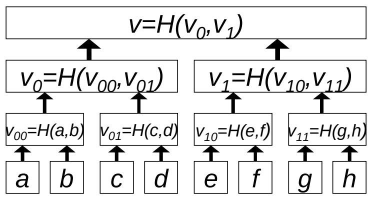
Merkle Hash Tree

- A method of committing to (by hashing together) n values, $x_1,\ldots,x_n$, such that
  - The result is a single hash value
  - For any x_i, it is possible to prove that it appeared in the original list, using a proof of length O(log n).



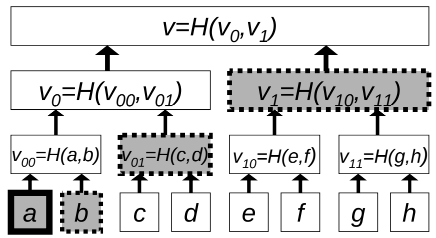
Merkle Hash Tree

- H is a collision intractable hash function
- Any change to a leaf results in a change to the root
- To sign the set of values it is sufficient to sign the root (a single signature instead of n).
- How do we verify that an element appeared in the signed set?



Verifying that *a* appears in the signed set

- Provide *a*'s leaf, and the siblings of the nodes in the path from *a* to the root. (O(log n) values)
- The verifier can use *H* to compute the values of the nodes in the path from the leaf to the root.
- It then compares the computed root to the signed value



Using hash trees to improve the overhead of CRS

- Originally (for a year long certificate)
  - the certificate includes $f^{365}(Y_0)$
  - On day *d*, certificate holder obtains $f^{365-d}(Y_0)$
  - The certificate receiver computes $f^{365}(Y_0)$ from $f^{365-d}(Y_0)$ by invoking *f()* *d* times.
- Slight improvement:
  - The CA assigns a different leaf for every day, constructs a hash tree, and signs the root.
  - On day *d*, it releases node *d* and the siblings of the path from it to the root.
  - This is the proof that the certificate is valid on day *d*
  - The overhead of verification is O(log 365).

Certificate Revocation Tree (CRT) [Kocher]

- A CRT is a hash tree with leaves corresponding to statements about ranges of certificates
  - Statements describe regions of certificate ids, in which only the smallest id is revoked.
    - For example, a leaf might read: “if 100 ≤ id <234, then cert is revoked iff id=100”.
  - Each certificate matches exactly one statement.
  - The statements are the leaves of a signed hash tree, ordered according to the ranges of certificate values.
  - To examine the state of a certificate we retrieve the statement for the corresponding region.
  - A single hash tree is used for all certs.

Certificate Revocation Tree (CRT)

- Preferred operation mode:
  - Every day the CA constructs an updated tree.
  - The CA signs a statement including the root of the tree and the date.
  - It is Alice’s responsibility to retrieve the leaf which shows that her certificate is valid, the route from this leaf to the root, and the CA’s signature of the root.
  - To prove the validity of her cert, Alice sends this information.
  - The receiver verifies the value in the leaf, the route to the tree, and the signature.
- Advantage:
  - a short proof for the status of a certificate.
  - The CA does not have to handle individual requests.
- Drawback: the entire hash tree must be updated daily.

Primality testing

- Why do we need primality testing?
  - Essentially all public key cryptographic algorithms use large prime numbers
  - We therefore need an algorithm for prime number generation
  - Suppose we have an algorithm “PrimalityTest” with a binary output.
  - We can generate random primes as follows

```
GeneratePrime(a,b)
  1. Choose random number x ∈ [a,b]
  2. If PrimalityTest(x) then output “x is
  prime”; otherwise goto line 1.
```

Density of prime numbers

- How long will GeneratePrime run?
- Let $\pi(n)$ specify number of primes ≤ n.
- Prime number theorem:
  - $\pi(n)$ goes to n / ln n as n goes to infinity.
- Pretty accurate even for small n (e.g. for $n=2^{30}$ it is off by 6%).
- Corollary: a random number in [1,n] is prime with probability 1/ln n. (e.g. for $n=2^{512}$, probability is *1/355*).
  - The GeneratePrime algorithm is expected to take ln n rounds.
  - If we skip even numbers, we cut running time by ½.

Primality testing

- Primality testing is a decision problem: "is x prime or composite?"
- Different than the search problem "find all prime factors of x".
- In this case, the decision problem has an efficient solution while the search problem does not.
- First algorithm: Trial division
  - Try to divide x by every prime integer smaller than $\sqrt{x}$ (sqrt(x)).
  - Infeasible for large x.

Fermat's test

- Fermat's theorem: if *p* is prime then for all $1 \le a < p$ it holds that $a^{p-1} = 1 \bmod p$.
- If we can find an *a* s.t $a^{x-1} \neq 1 \bmod x$, *x* is surely composite.
  - Surprisingly, the converse is almost always true, and for a large percentage of the choices of *a*.
  - Suppose we check only for *a=2*.
    - If 2^{x-1} != 1 mod x
      - Then return COMPOSITE /for sure
      - Otherwise, return PRIME /we hope
  - How accurate is this program?

Fermat's test

- Surprisingly, this test is almost always right
  - Wrong for only 22 values of x smaller than 100,000
  - Probability of error goes down to 0 as x grows
    - For |x|=512 bits, probability of error is $< 10^{-20} \approx 2^{-66}$
    - For |x|=1024 bits, probability of error is $< 10^{-41} \approx 2^{-136}$
- The test is therefore sufficient for randomly chosen candidate primes
- But we need a better test if x is not chosen at random
- Cannot eliminate errors by checking for bases ≠ 2
  - *x* is a Charmichael number if it is composite, but $a^{x-1} = 1 \bmod x$ for all $1 \le a < x$.
  - There are infinitely many Charmichael numbers
  - But they are rare

Miller-Rabin test

- Works for all numbers (even Charmichael numbers).
  - Checks several randomly chosen bases a
  - If it finds out that $a^{x-1} = 1$ mod x, it checks whether the process found a nontrivial root of 1 ($\neq 1,-1$). If so, it outputs COMPOSITE.

The Miller-Rabin test:

1. Write $x-1=2^c r$ for an odd r. set comp=0.
2. For i=1 to T
   - Pick random a $\in$ [1,x-1]. If gcd(a,x)> 1 set comp=1.
   - Compute $y_0=a^r$ mod x, $y_i=(y_{i-1})^2$ mod x for $i=1..c$. If $y_c \neq 1$, or $\exists i$, $y_i=1$, $y_{i-1} \neq \pm 1$, set comp=1.
3. If comp=1 return PRIME, else COMPOSITE.

Miller-Rabin test

- Possible values for the sequence $y_0=a^r$, $y_1=a^{2r}\ldots$ $y_c=a^{x-1}$.
  - <…,d>, where $d \neq 1$, decide COMPOSITE.
  - <1,1,…,1>, decide PRIME.
  - <..,-1,1,..,1>, decide PRIME.
  - <…,d,1,…,1>, where $d \neq \pm 1$, decide COMPOSITE.
  - For a composite number x, we denote a base a as a non-witness if it results in the output being “PRIME”.
- Lemma: if x is an odd composite number then the number of non-witnesses is at most $x/4$.
- Therefore, for any odd integer x, T trials give the wrong answer with probability < $(1/4)^T$.