Cryptography and security: Quick reference

vishvAs vAsuki

June 16, 2012

Based on Brent Waters's course, [1] and [2].

1 Notation

wnnp: with non negligible probability. p assumed to be prime.
Security with auxiliary input: Security is not broken even if functions of SK are revealed.

2 Themes

Communication in the presence of an adversary. Encryption; efficiency of encryption: important due to pervasive communication; Batch verification. Authentication. Key distribution. Access control. Crypto-analysis: Break hardness assumptions used in cryptography. Make better proofs of security.
Proving the security of protocols. Code obfuscation: Hide your intent. 0 knowledge proofs.
Oblivious transfer: evaluate a function on many agents' sercrets, without anyone knowing anyone's secret.

2.1 Network protocol security

The security vs functionality tension. How do you design and prove the security of protocols for various purposes, wrt various attacks?

2.2 Applications

Mainly useful for secretive corporations, governments and citizens who care for privacy.

2.3 Characterization of research effort

2.3.1 General strategy

Finding scenarios with security problems. Informal/ creative thought, understand threat model, formalize security definition, describe solution, prove security using widely accepted assumption.
Typical proof: assume some attacker A can break security; then given access to A, you make an algorithm to violate assumption. Easy to make mistake: Unless you write down proof in complete detail, implicit unwarranted assumptions about A creep in: Use propositional logic to state the thing to be proved.

3 Secure communcation over insecure channel

3.1 Problem

A and B communicate, E evesdrops. E should not know what is communicated, and E should not be able to cause miscommunication between A and B.

3.1.1 Unbounded adversary

(Shannon) Theory of perfect secrecy. Adversary assumed to have unlimited computational resources. Secure encryption system exists only if $|S|$ is as large as $|m|$.

3.1.1.1 One time pad

$E(pad, m) = m \oplus pad$; $D(pad, c) = c \oplus pad$. Unbreakable even by computationally unbounded adversary: Modern cryptography abandons this. $\forall m, c : Pr_{pad}(E(pad, m) = c) = 2^{-n}$: so perfectly secret. But not good for 2 messages: $E(pad, m_1) \oplus E(pad, m_2)$ reveals common bits.

3.1.2 Bounded adversary

Often parametrized by security parameter k.

3.1.3 Passive adversary

Possibly randomized algorithm which runs in poly(k) time, has passive access to everything on the insecure channel. May know probability distribution over messages.
Is said to break cryptosystem if it succeeds with non negligible probability: $\frac{1}{poly(k)}$.

3.1.4 Active adversary

Passive adversary with extra powers. Can alter or stop messages, request polynomial number of cyphertexts to be decyphered.
A common active adversary attack is the replay attack.

3.1.5 Collusion attacks

Multiple adversaries share info and attack.

3.2 Proving security/ vulnerability

3.2.1 Security from evesdropping

Prob of predicting m given c = Prob of predicting m without c: seeing c gives no 'information'.

3.2.2 Specify scheme

Describe Setup(l), encrypt, decrypt, keygen etc.. algs.

3.2.3 General strategy

Show security of B based on security of A: $Secure(A) \implies Secure(B) \equiv attackable(B) \implies attackable(A)$: The latter form is more convenient to prove.

3.2.3.1 Make security game

Take polytime B attacker, call it Z. Make B challenger/ A attacker, call it Y. Take A challenger, call it X. Visualization: use boxes and arrows to see how Y uses Z to respond to X's challenge.

3.2.3.2 Show success of attack wnnp

Security parameter l: All probabilities and running time are in terms of this. Use cases to analyze probability of success of attack.

3.2.4 Show equivalence of 2 models of security

Show that a successful attacker in one can be used to build a successful attacker in another.

3.2.5 Using hardness assumptions to prove security

Take hardness assumption H. Use this as A.

3.2.5.1 Tightness of assumption

Cryptosystem B, assumption A. Show that you can break A given attacker for B. Show that you can break B given attacker for A.

3.2.6 Security from weaker adversaries

Sometimes, proof of security against CPA is not known. So, security is proved against weaker adversaries. Eg: Security in random oracle model, selective security etc..
A stepping stone in finding proof of security against unhindered adversary.

3.3 Private key encryption

Communicators A and B; Encryption and decryption algorithms E, D; common secret key S; adversary may know (E, D), but not S; cleartext or plaintext message m; ciphertext c = E(m,S).

3.3.1 Substitution cipher

S = the permutation f. Easy to break by adversary who sees moderately many ciphertexts. [**Find proof**]

3.4 Public key cryptography

Aka assymetric key cryptography.

3.4.1 Weak definition

Uses trapdoor one way functions. Setup(l) = (PK, SK). Weak defn: E(m, PK) = c; D(c, SK) = m: If A transmits only 2 messages, Attacker could encrypt both messages and say what is being transmitted.

3.4.2 Strong defnition: blinding messages

So, random r is chosen, m is blinded using r to get m', $E(m, r, PK) = c$. D(c, SK) = m', r; thence unblind to get m.
Blinding and unblinding is often done by computing a blinding factor (bf). Agents agree on bf in a way similar to key exchange protocols.

3.4.3 Hybrid cryptography

In practice, this is used, rather than pure public key cryptography as it is slow. Public key crypto is used only to share a common secret 'session key' S. S is then used to actually encrypt messages.

3.4.4 Semantic security: chosen plaintext attack (CPA)

Take challenger C and attacker A.
C sends PK to A. A sends plaintexts m_0 and m_1 to C. C picks $g \in_U \{0,1\}$ and sends $c = E(m_g, r, PK)$. A surmises and returns g.
If A has non negligible advantage over random guessing, the crypto scheme is broken by A.

3.4.5 Semantic security from CCA

Same as CPA security game, with extra powers to the attacker: The attacker A can ask for decryption of poly num of cyphertexts in two phases: one before challenger C sends $c = E(m_g, r, PK)$, and one afterwards. In second phase, A cannot demand decryption of c. If given an invalide cyphertext, C responds with 'invalid cyphertext' or $\perp$ message.

3.4.6 Hybrid proofs

Eg: two public key schemes: X1, X2; new PK scheme X encrypts m to users using both; show $secure(X1) \wedge secure(X2) \implies secure(X)$. So show $\neg s(X) \implies \neg s(X1) \vee \neg s(X2)$: Consider CPA game where attacker B knows $\{m_0, m_1\}$, X challenger A picks random bit g, encrypts m_g; gives B $c_0 = enc(m_g, X1), c_1 = enc(m_g, X2)$; B guesses g'; So $\neg s(X) \equiv Pr(g' = g) \geq 2^{-1} + \epsilon$. Take $p_1 = Pr(g' = 1 | g = 1), p_0 = Pr(g' = 1 | g = 0)$; imagine hybrid case: $p_h = Pr(g' = 1 | enc(m_0, X1), enc(m_1, X2))$. $\neg s(X) \equiv Pr(g' = g) \geq 2^{-1} + \epsilon \equiv 2\epsilon \leq |p_1 - p_0| \leq |p_1 - p_h| + |p_0 - p_h| \equiv |p_1 - p_h| \geq \epsilon \vee |p_0 - p_h| \geq \epsilon \equiv \neg s(X1) \vee \neg s(X2)$.

3.4.7 RSA

Choose random N=pq with p, q large primes : use rand alg; pick PK: exponent $e \in \{1, ..\phi(N) - 1\}$ coprime with $\phi(N)$; choose SK exp d to satisfy $de = 1 \bmod \phi(N)$; PK: (N, e); SK: (N, d). Message of length m; encrypt: $c = m^e \mod N$: exponentiation by squaring; decrypt: $c^d \mod N$.
Security from RSA hardness assumption. Vulnerable to CCA.
This is the fastest PK scheme.

3.4.8 ElGamal

$PK = (g, g^y)$. SK = y. Encr: Pick random r, make bf: $(g^y)^r$. $c = g^r, m.g^{yr}$. Decr: Recover $(g^r)^y$, thence decrypt. Agreement on bf similar to DH key exchange.

3.4.8.1 Security from CPA if DDH hard

If DDH is hard, ElGamal is secure: Consider DDH Challenger DC, DDH attacker DA, El Gamal attacker A. DC picks random t, sets $T = g^{ab}$ if t=1. DC

gives g, g^a, g^b, T to DA. DA gives $PK = g, g^a$ to A. A gives m_0 and m_1 to DA. DA picks random b and gives A $c = T, m_b.T$. If T is a valid bf, A returns g wnnp. Thence, DA can defeat DC wnnp: analyze probabilities of success for the cases where t=0 and t=1.

3.4.8.2 Vulnerability when DDH broken

Can use any DDH atatcker to break ElGamal: show with a security game.
True if there are efficiently computable bilinear maps.

3.4.8.3 CCA vulnerability

Attacker demands decryption of $c' = g^{r+r'}, hm_b.g^{y(r+r')}$, thence identifies m_b: cyphertext for hm_b used as some challengers may refuse to decrypt a previously sent msg.

3.4.9 CCA secure scheme from IBE scheme

PK = PP; SK = MSK, SigSK (Signature SK), SigPK. Enc(m) : choose random id, find
$CT = Enc_{IBE}(PP, ID, m)$, make CT, id, sigSK(CT).
Dec: if $sigPK(sigSK(CT)) \neq CT$ abort; else get $Keygen(MSK, id) = SK_{id}$; do $Dec_{IBE}(CT, SK_{id}) = m$.

3.4.9.1 Speed

1 pairing costs approximately as much as 8 large exponentiations.

3.4.10 DLA based cryptosystem

Setup(l): $SK = (x, y)$.
$PK = (g, u = g^x, v = g^y)$. Encrypt by selecting random a and b. Come to agreement on BF: v^{a+b}.

3.4.11 Applications

Public key cryptography is highly secure. But, it is often slower compared to symmetric key encryption.
SSL uses public key cryptography for key exchange and symmetric encryption for privacy.

3.5 Key exchange

3.5.1 Diffie Hellman (DH)

A chooses generator g, prime p, random $a \in G$. Sends $(g, p, g^a \mod p)$ to B. B picks random b, sends $g^b \mod p$. Both A and B now find secrect key $S = g^{ab}$

mod p.

3.5.2 3 party key exchange

A, B, C pick x, y, z; have bilinear map $e : G \times G \to G_T$, generator $g \in G$; publish g^x, g^y, g^z; agree upon SK $= e(g,g)^{xyz} = e(g^x, g^y)^z$.

3.6 Access control

Server based vs encryption based.

3.6.1 Role based access control

Functional encryption: Specify access policy (a bool formula) as part of ct.

3.6.2 Sharing a secret

Want people with 2 attributes to access something: Share secret info $m \in G$ between 2 roles: Turn policy into a infix boolean tree: When you encounter $\wedge$, split m into r, m-r. Subject to collusion attack: Solve by initially encrypting with requestor's PK.

3.7 Broadcast encryption

Bandwidth limited. Eg: Direct TV, radio, GPS.
Like IBE. Setup(l,n): $PP, SK_1..SK_n$.
$Enc(PP, m, S \subseteq [n]) : c$. $Dec(c, i, SK_i, S \subseteq [n]) = m$ iff $i \in S$.
Want collusion resistance. t-collusion resistance: $|c| \geq t^2 \log n$ [**Find proof**].

3.7.1 CPA Security

Challenger A, attacker B. A to B: PP; B to A: keygen requests, get $\{SK_i\}$; B to A: target set S', chosen messages m_1, m_2; A picks random b, returns $enc(PP, m_b, S')$. B can encrypt arbit m himself.
Static security: announce S' first.

3.7.2 BB IBE based scheme

(BGW) Setup(l,n): Pick random a, g, $\{u_1..u_n\}$, $\{r_1..r_n\}$. $PP : e(g,g)^a, \{u_i\}$.
$SK_i : g^a u_i^{r_i}, g^{r_i}, \forall_{j \neq i} u_j^{r_i}$.
$Enc(PP, m, S) : me(g,g)^{as}, g^s, \prod_{i \in S} u_i^s$ for random s: like BB IBE enc
to $\prod_{i \in S} u_i^s$: a SK for every S.
Dec: Want to find bf: $e(g,g)^{as}$; so find $e(g^s, g^a u_i^{r_i})$, find $e(g^{r_i}, \prod_{i \in S} u_i^s) = \prod_{i \in S} e(g^{r_i}, u_i^s)$, divide by $e(g^s, \prod_{j \in S, j \neq i} u_j^{r_i})$.

3.7.3 Security against q-BDHE

A q-BDHE challenger, B q-BDHE attacker, C BGW attacker.
A to B: $g, h, g^b, g^{b^2}, ..g^{b^q}, g^{b^{q+2}}..g^{b^{2a}}$: no $g^{b^{q+1}}$; see if $T \stackrel{?}{=} e(g,h)^{b^{q+1}}$.
C to B: S'. B sets $h = g^s, a = b^{q+1}$; picks (y_i), sets $\forall i \in S' : u_i = g^{y_i}$, other i: $u_i = g^{b^i} g^{y_i}$. B answers keygen reqs: need find $g^a u_i^{r_i} = g^{b^{n+1}} (g^{b^i} g^{y_i})^{r_i}$, don't know g^a, so try cancellation by picking $r_i = -b^{n+1-i}$; actually, to make r_i look random, must add some x_i. C to B: m_1, m_2. B to C: picks m_b, sends $m_b T, g^s, \prod_{i \in S'} h^{y_i}$.

3.7.4 Piracy of broadcast system

DRM impossibility argument: Can't protect content from leaking out. Can only protect original broadcast stream. Traitor tracing: Who pirated the original stream?

4 Match public keys with identity

Usually trusted key-servers are used.

4.1 Identity based encryption (IBE)

IBE Authority (Auth) publishes public parameters (PP), has master secret key (MSK). $Setup(l) \rightarrow (PP, MSK)$. c = E(PP, ID, m). $SK_{id} = K(MSK, ID)$. $m = D(SK, c, PP)$.

4.1.1 Security

4.1.1.1 Semantic security under CPA

Challenger C, attacker A. C sends PP to A. A sends C $\{ID_i\}$. C returns $\{SK_i\}$. A chooses target ID^*, sends it to C. A sends C m_0, m_1. C randomly picks g and sends $c = E(PP, ID^*, m_g)$. A sends C $\{ID_i\}$. C returns $\{SK_i\}$. If A returns correct g wnnp advantage over random guessing, attack is successful.

4.1.1.2 Security under the random oracle model

If Hash fn H is used. Adversary assumed not to have access to H code, but only oracle access to H used in the protocol. H returns a random group element when queried.
Used in SSL, PGP security proofs.

4.1.1.3 Selective security under CPA

Adversary must choose target before seeing PP.
Can be selectively secure without full security: take any fully secure scheme X with algs Setup, KeyGen, Encrypt, Decrypt. Make selectively secure but

not fully secure scheme Y by saying keygen'(id) = keygen(id) if id = tid, and keygen(oid).

4.1.1.4 Main challenges in proofs

Make keys for $id \neq tid$; use attacker's response to break assumptions. Usually met in Setup, hash fn oracle.

4.1.2 Applications

If recipient is not online, can't get public key directly from him. IBE: No need to look up public key. Also, auth can grant SK at a future date, enabling messages which can be opened at a future date,

4.1.3 Disadvantages

Auth can decrypt anything. If CA is compromised, it will be noticed. But if auth is compromised, this may not happen.

4.1.4 Boneh Franklin (BF) system

Setup(l): $MSK = y \in Z_p$; PP: $g, g^y, H : \{0,1\}^n \rightarrow G$. Collision resistant hash fn H can handle long ID's. Bilinear map $e : Z_p \times Z_p \rightarrow G$ with order p. Encrypt: Pick $m \in G$, find $e(H(ID), g^y)^s$; $c_1 = g^s$, bf $e(H(ID), g)^{ys}$. Keygen: $SK_{id} = H(id)^y$. Decrypt: $e(SK_{ID}, c_1)$ to get bf.

4.1.4.1 Security against CPA under Random Oracle model

DBDH challenger A, DBDH attacker/ IBE challenger B, IBE attacker C. C makes q oracle queries. Before attack starts, B guesses C's target id tid, fixes H so that $H(tid) = g^b$, random element in G otherwise. B aborts if C queries H(tid). Account for this probability in proof.

4.1.5 Boneh Boyen

Setup: Pick $g, h \in G; a, b \in Z_p$. Bilinear map e. $PP : g, h, u = g^a, e(g,g)^{ab}$. $f(id) = u^{id}h$. $MSK : g^{ab}$.
Randomized keygen: new key each time: Pick $r \in Z_p$; $k_1 = g^{ab}.f(id)^r$; $k_2 = g^r$: like encrypting MSK. If r were not random, could query 2 SK's, divide, find u^r and then find anyone's SK.
Enc(PP, M, ID): pick $s \in Z_p$, $c_1 = g^s$, $c_2 = f(ID)^s$, $c_0 = m.(e(g,g)^{ab})^s$.
Dec:
$e(k^1, c_1) = e(g^{ab}.f(id)^r, g^s) = e(g^{ab}, g^s)e(f(id)^r, g^s) = e(g,g)^{abs}e(k_2, c_2)$: bf for bf!.

4.1.5.1 Selective security under CPA

DBDH challenger A, DBDH attacker/ IBE challenger B, IBE attacker C. A: g, g^a, g^b, g^s, T. C: Attacking tid. B. Setup: $g, u = g^a, e(g,g)^{ab}, h = g^{-a(tid)}g^y$. So B's view of f: $f(tid) = g^y, f(id) = g^{a(id-tid)}g^y$; but in C's view f is as random as it would be if u and h were picked randomly from G: C is guaranteed to succeed wnnp only in such a case.
Keygen: pick $r = \frac{-b}{id-tid}$, so $k_2 = g^{\frac{-b}{id-tid}}, k_1 = g^{\frac{-y}{id-tid}}$. For tid, $k_1 = k_2 = 1$. But C is not guaranteed to work in such a distribution: so rerandomize the key: pick rand $t \in Z_p$; set r := r+t; get $k_1 := f(id)^t k_1, k_2 := g^t k_{id}$. [**Check**]
B sends cyphertext: $c_0 = m_g T, c_1 = g^c, c_2 = g^{cy} = f(tid)^c$.

4.1.6 Waters cryptosystem

Instead of guessing tid as in BF, guess 1/q of the space of id's as possible targets. Setup: $g, e(g,g)^{ab}, h, u_1, .., u_n$; $f(id) = h\prod u_i^{id_i}$ where id_i = ith bit of id.
Fully secure. Selective id proof led us in the right direction.

4.1.7 Non pairing based IBE cryptosystems

Based on learning parity with noise hardness assumption by Vaikuntanathan et al.

5 Authentication

5.1 Using assymetric key cryptography

One simple way of authentication is to encrypt and return a challenge message using one's secret key.

5.2 Using a password

This is very common - eg: it is used over to log into terminals.
To beat snooping, passwords are transmitted over a secure channel (eg: ssh vs rsh).

5.2.1 One-time passwords

Passwords may be spoofed - even over a secure channel by an adversary who jumps into a session and replays the password message to, for example change the password. To guard against this, a one-time password can be used. These are often based on a one way function f with an initial point p, which can be used to generate a sequence $f^i(p)$. To do this, people are often provided with a special physical device.

5.3 Signing public messages, and their authentication

setup(l): SK, PK, s = sign(m, SK), verify(m, s, PK). Algs use hash fn: H.
Not all PK cryptosystems can be converted to signature schemes: security proof can fail.

5.3.1 Existantial unforgeability under chosen message attack

Challenger (A) vs attacker (B) game. A to B: PK. B gets many msgs $\{m_i\}$ signed by A. Finally, B forges a $m^* \notin \{m_i\}$.

5.3.1.1 Main challenges in proofs

A, B, C game. Generation of signatures by B; Use of C's forgery to break A.

5.3.1.2 Weak signature scheme

B tells A list $\{m_i\}$ he wants signed before B sees PK.

5.3.1.3 One-time signature scheme

Attacker allowed to make only one signing query. Maybe after seeing PK.

5.3.1.4 Unforgability using weak and one-time signatures W, O

setup(l): Run $setup_o(l)$ poly(l) number of times. Get vector $((OSK_i, OVK_i))$. Run $setup_w(l)$ and get VK_w, SK_w. Sign all $\{OVK_i\}$ with SK_w to get the vector $(SOVK_i)$. Set $PK = PK_w$.
sign(m): Pick $(OSK_i, OVK_i, SOVK_i)$ triple
not used eariler. Get $s_1 = sign_o(m, OSK_i)$. Set $s = s_1, SOVK_i, OVK_i$.
verify(m, s): $verify_w(SOVK_i, OVK_i, VK_w) \stackrel{?}{=} 1$ and $verify_o(s_1, m, OVK_i) \stackrel{?}{=} 1$.

5.3.2 Schnorr signature

Setup: Take G of prime order p; Pick g, a; PK: g, g^a. SK: a. $H : \{0,1\}^* \times G \to Z_p$. Sign(m, SK): Pick random g^k; $e = H(m, g^k)$; t = k + ae mod p; $s = (g^k, t)$.
Verify(m, s, PK): Know e, g, g^a, g^k; chk $g^t \stackrel{?}{=} g^k (g^a)^e$.

5.3.2.1 Unforgeability under random oracle assumption

A: DL challenger; B: DL attacker; C: Schnorr attacker. B gets g, g^a from A, passes it on to C as PK. When C asks for sign(m, SK), B picks random e; want t = k + ae mod p, pick k = r - ae; fill up random oracle table $H(m, g^k) = e$. B knows random bits used by C. When C forges, it uses some query $H(m, g^k)$; random oracle gives e_1, set up by B for g^{k_1}: so B, given the forgery, knows $t_1 = k_1 + ae_1$. Then B rewinds C to the same point and instead supplies e_2, finds $t_2 = k_1 + ae_2$, finds a.

5.3.3 RSA signature

Use RSA SK to sign: $H(m)^d \mod n = \text{s}$. Use RSA PK to verify: $s^e \mod n \stackrel{?}{=} H(m)$. e chosen to be small: like 3: faster verification. Can't have d too small: attack can try all small numbers.

5.3.3.1 Unforgeability under random oracle assumption

Not based on DL. A: RSA assumption challenger; B: RSA attacker; C: RSA forger. A gives B: N, e, h^e. Game mostly same as Schnorr. When B must sign m, it picks y, sets $H(m) = y^e$, returns y. When C forges using query H(m), it is given h^e.

5.3.4 Signature from GHR

Setup: PK: N = pq, u, w; SK: p, q. Sign(m): pick prime $e < \phi(N)$; $s = (u^m w)^{e^{-1}}, e$. Verify: $s^e \stackrel{?}{=} (u^m w)$.

5.3.4.1 Unforgeability under weak signauture scheme

A: Flexible RSA challenger; B: FRSA attacker; C: GHR forger. B gives A set $\{m_i\}$; A replies with $\{s_i\}$ with $\{e_i\}$. A gives B PK depending on 2 cases. Case 1: Guess that forgery on m_k; pick $g = h^{m_j - m_k}$, set $w = g^{\prod_i e_i} h^{-m_k \prod_{i \neq k} e_i}$, $u = h^{\prod_{i \neq k} e_i}$; answer sig query for m_j with $h^{\Delta m \prod_{i \neq k, j} e_i} g^{\prod_{i \neq j} e_i}$; get forgery $s = g^{\prod_i e_i / e'}$; use KS with $x = h, a = \Delta m \prod_{i \neq k} e_i, y = s, b = e_k$ [**Check**]. Case 2: Pick $a, w = h^{\prod e_i}, u = h^{a \prod e_i}$ [**Check**]. [**Incomplete**]
KS Lemma: If $x^a = y^b \mod p; gcd(a, b) = 1$, can find z: $z^b = x$ [**Find proof**].

5.3.5 Signature from BF IBE

(BLS) BF IBE uses full domain fn H. SK: y. PK: g, g^y. Encrypt: $Keygen(m) = H(m)^y = s$. Verify: $e(s, g) \stackrel{?}{=} e(H(m), g^y)$.

5.3.5.1 Unforgeability using random oracle

A: CDH challenger, B: BLS challenger, C: BLS attacker. A to B: g, g^a, g^b. B: SK = b (not known to b), PK:g, g^b. B guesses that C forges kth query to random oracle, sets $H(m_k) = g^a$; for other i, picks random n, sets $H(m_i) = g^n$. B replies to sign requests by picking x, setting $H(m) = g^x$, g^{xa}. B uses forgery to break CDH.

5.3.6 Signature from any IBE

Setup(l): Get PP, MSK. Set PK = PP; SK = MSK. Sign(m): s = Keygen(H(m), MSK). Verify(m, s): Pick some other msg n. Get c = Encrypt(n, PP, H(m)), then get n' = decrypt(c, s), do $n \stackrel{?}{=} n'$.

5.3.7 Aggregate signatures

Eg: petition signing, BGP. Verify n msgs with n PK's: $m_1..m_n s_{agg}$. Algorithms: Setup: (PK, SK); $Sign(SK_i, m) = \sigma_i, m$; $aggregate(s_{1..k}, \sigma_{k+1}) = s_{k+1}$:
ordering should not matter; $verify(\{PK_i\}, m_1, ..m_n, s_{1..n})$.

5.3.7.1 Unforgability of aggregate signatures

A: Agg sign challenger; B: attacker. A to B: PK'. B gets many messages signed using SK'. B forges; forgery:
$\{m_i, m'\}, \{PK_i, PK'\}, s$: B not told $sgn(m')$ earlier.

5.3.7.2 Aggregate signature using BLS

$s_{agg} = \prod H(m_i)^{y_i}$ would fail: B can tell A: $s_{agg} = H(m')^{y_1} H(m')^{-y_1} = 1$. Instead use: $s_{agg} = \prod H(m_i, PK)^{y_i}$.

5.3.8 Applications

Proof of storage.

5.3.8.1 Certificate chains for PK's

PK often accompanied by certificate: something signed with the SK of certificate authority (CA). This signature often accompanied by another certified PK. This continues recursively until the trusted root CA's signature.

5.4 Proof of storage

Owner stores something on storer; who proves storage using a protocol. Owner calls Store(D) $\rightarrow$ Verification Key VK, Storage File SF; owner keeps VK, gives storer SF. Audit interactions: prove(SF) $\leftrightarrow$ verify(VK). Correctness condition: verify(VK) convinced wnnp iff $\exists$ extractor: E(p) = D.
Performance measures: Audit speed, bandwidth between owner and storer during audit, verifier storage, prover storage.
Store(D): SF = D; $VK = \{(h_i = H(K_i, D), K_i)\}$. Disadvantage: Verification works n times. Usually use random oracles for extraction.

5.4.1 Based on Erasure code

Code expands m blocks (D) to n blocks; if ye have n/2 blocks, can recover D. SF is the n blocks, with signatures $\{sgn(block_i, i)\}$; VK = signature keys. Verify() wants to check that storer has n/2 blocks whp: Verify asks for k

random blocks; storer gives these blocks; verifier checks signature in response; if storer has $< n/2$ blocks, Pr(Audit succeeds) $\leq 2^{-k}$.

6 Other problems

6.1 Code obfuscation

Hide the intent of the code. Security with auxiliary input useful here. $M \rightarrow O(M)$ with polynomial blowup in size, run-time. M, O(M) compute the same function: or maybe approximately. Virtual black box property: $\forall$ polytime algs A, $\exists$ simulator S^M with black box access to M: $|Pr(A(O(M)) = 1) - Pr(S^M(1^{|M|}) = 1)| \leq \epsilon$: $1^{|M|}$ bounds run-time; whatever property of M A can grok by looking at the code, S can grok just by looking at its behavior.
If you can exactly learn C, $c \in C$ can't be obfuscated.

6.1.1 Point functionss

Eg: password, cd key. Point fn can be obfuscated. [**Find proof**]

6.2 0 knowledge proofs

Prover P, Verifier V. P proves statement s to P wihtout giving away the proof. Eg: convince V that N=pq, a product of exactly 2 primes without giving away p, q. Useful in many crypto protocols.

6.2.1 0 knowledge proofs of membership ($x \in L?$) for NP complete languages

Take graph G = (V, E), $|E| = m$; P wants to show V that $G \in 3 - COLOR$; P knows valid coloring C. P commits to C by sending encrypted $enc(C(x)) \forall x \in V$ (example of cryptographic commitment protocol); Let V pick any $e = (u, v) \in E$; P reveals colors of u, v in C by sending keys to only $enc(C(u)),\ enc(C(v))$; P permutes colors in C; the cycle repeats. If C invalid, P will will fool V with prob $\leq (1 - m^{-1})$; so after m^2 repeatitions, P is very unlikely to have been fooled.
3-COLOR is NP complete; so can translate any NP complete language membership problem to this and use 0-knowledge prover.

6.3 Oblivious transfer OT

Sender $S \to R$; S has information p,q. R wants some info x = p or q from S, x must be oblivious to S. [**Incomplete**]

7 Cryptographic primitives

7.1 Collision resistant hash function

A hash function where it is hard for an adversary to find y: h(x) = h(y), given h(x) and y.

7.2 Strength of hardness assumptions

If violation of hardness assumption A implies violation of assumption B, A is weaker than B. Weaker assumptions are preferred.

7.2.1 To show weakness $A < B$

Take poly time alg to break A and make poly time alg to break B whp.

7.2.2 Hierarchy of strength

$DL < CDH < DDH < CDH$.

7.3 Candidate one way functions

Easy to compute, hard (takes superpolynomial time) to invert whp. Often from computational number theory. Useful in key exchange.

7.4 One way functions based on Factoring

$f : (x, y) \to xy$. Believed to be hard.

7.4.1 Hard core predicate h for one way functions F

No prob. polytime alg to predict output of h using F significantly better than random guessing, over U.

7.5 Group theoretic functions

Group G, generator g. In practice, $|G| \approx 2^{160}$.

7.5.1 Number theoretic groups

For alg for finding large primes, see number theory ref. Can efficiently find generators [**Find proof**].

7.5.2 Elliptic curve groups

See Topology ref.
The discrete log problem on elliptic curve groups is believed to be more difficult than in the underlying finite field's multiplicative group. So shorter keys yield same security.

7.5.3 Discrete logarithm (DL)

$x \in Z_p^*$, g its generator, $DLog_{p,g}(a) :=$ least power $i \in \{0, ..p-2\}$: $g^i = a$.
Breaking: So, given a, p, g: find i.

7.5.4 Computational Diffie Hellman (CDH)

Pick a, b randomly from G.
Breaking: Given g^a, g^b, g, output g^{ab}.

7.5.5 Decisional Diffie Hellman

Pick a, b from G. Pick $d \in_U \{0,1\}$. If d = 0, output $T = g^{ab}$, else output $T \in_U G$.
Breaking: Given the Diffie Hellman tuple $\left(g^a, g^b, g, T\right)$, guess d. Do this significantly better than random guessing.
Not secure if bilinear map efficiently computable.

7.5.6 Decisional linear assumption (DLA)

Decisional linear problem: Given group G of prime order p and elements g, u, v, g^a, u^b, distinguish $T = v^{a+b}$ from a random number. T is chosen to be v^{a+b} based on a random bit s.

7.6 Group with efficiently computable bilinear map e

Now adversary has access to p.
CDH assumed to be hard.
DDH is no longer hard: Given g, g^a, g^b, T, check if $e(g,T) = e(g^a, g^b)$.

7.6.1 Bilinear DDH (BDDH)

Extends DDH.
Breaking: Do this significantly better than random guessing:
Given g, g^a, g^b, g^c, distinguish g^{abc} from random T wnnp.

7.6.2 q-BDHE

Given $g, h, g^a, g^{a^2}, ..g^{a^q}, g^{a^{q+2}}..g^{a^{2a}}$:
no $g^{a^{q+1}}$; distinguish $e(g,h)^{a^{q+1}}$ from random r. A different assumption for each q!
Can prove security (if $P \neq NP$) under generic group model: only certain ops (pairing, arithmatic ..) allowed; oracle access to pairing fn.

7.7 Information theoretic one way functions

Decoding random (n, k) linear codes. See Information and coding theory ref for details.

7.8 Assumptions from learning theory

Learning subspace with noise assumption. Weaker than LPN.
Learning partity with noise (LPN) assumed to be hard.

7.9 Lattice based cryptography

Does not assume hardness of factoring.
[**Incomplete**]

7.10 Trapdoor one way function

One way function f which also has some trapdoor used to invert f.

7.10.1 RSA hardness assumption

p,q prime. N=pq. Given $y = x^e \mod N$, N, e, hard to find $y^{\frac{1}{e}}$. Hard to find d given e and N by factoring N=pq, then finding $\phi(N)$ and then finding d by Euclid's Algorithm.

7.10.1.1 Discrete cube root

Special case where e = 3.

7.10.1.2 Circuits to compute RSA fn: Iterated products problem

Multiply n n-bit numbers mod N. Computable by polynomial sized ckts of depth $O(\log n)$.
Finding $x^e \mod N$ efficient with poly(n) sized boolean ckts which use repeated squaring and multiplying the right squares.
Similarly, can compute ith output bit of BBS pseudorandom generator (see randomized algorithms ref).

7.10.1.3 Strong/ Flexible RSA assumption

Got N = pq, h; Find some $e, h^{1/e}$.
Stronger than RSA ass.

Bibliography

[1] Shafi Goldwasser and Mihir Bellare. *Lecture Notes on Cryptography.* Internet, 2008.

[2] Michael J Kearns and Umesh V Vazirani. *An introduction to computational learning theory.* MIT Press, Cambridge, MA, USA, 1994.