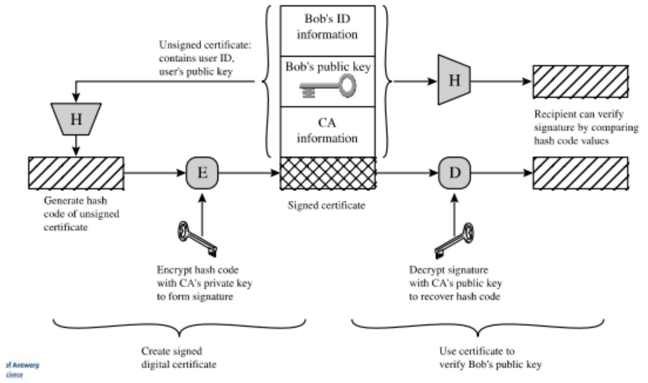
PART 2: Caesar cipher: Replace each letter in the alphabet with the letter k places further. General substitution: each plain text letter is assigned a unique random letter from the alphabet. Frequency attack: track letters based on most used letters in the alphabet. Vignere: Use all 26 mono-alphabetic general Caesar cipher substitution rules. A key represents the order in which the 26 substitution rules are used to encode subsequent plaintext letters. Encr: ci = (pi + k_imodM) mod N Decr: pi = (ci - k_imodM) mod N One Time Pad: M, C, K element of {0,1)^n. Encr: C = M xor K, M = C xor K RC4: PRG (pseudo random generator) maintains an internal state S of 256 bytes -> init S, S[0] = 0, ..., S[255] = 255 -> Temp vect T, if T.length = K.length => T = K else K copied on T, and then K repeated till T is filled with K -> j = 0 for i = 0 to 255 do: j = (j + S[i] + T[i]) mod 256; swap(S[i], S[j]) -> Stream gen, i, j =0 while true: (i+1) mod 256; j = (j + S[i]) mod 256: swap(S[i], S[j]); t = (S[i] + S[j]) mod 256; k = S[t]; Encr: Xor the value k with next byte of plaintext Decr: Xor the value k with next byte of the ciphertext. Salsa- ChaCha20: PRG is bade on add-rotate-XOR, 32-byte addition, bitwise addition, fixed binary rotation. Input: 256-bit key, 64-bit nonce, 64-bit counter. Internal state = {C1, K1, K2, K3; K4, C2, N1, N2; P1, P2, C3, K5; K6, K7, K8, C4}, C = 128-bit constant "expand 32-byte k", K = 256-bit key, N = 64-bit Nonce, P = 64-bit position (counter). Little endian format, so split and convert to little endian. QR() round function: (y0, y1, y2, y3) = QR(x0, x1, x2, x3) => y1 = x1 xor ((x0 + x3)<<<7), y2 = x2 xor ((y1 + x0)<<<9), y3 = x3 xor ((y2 + y1)<<<13), y0 = x0 xor ((y3 + y2)<<<18). Key reuse attack: c1 = m1 xor PRG(k) and c2 = m2 xor PRG(k) => c1 xor c2 = m1 xor m2 => dictionary attack m1, m2. Stream ciphers are malleable: E(k,m) xor t = m xor S(k) xor t = E(k, m xor t) => bit flipping attack. -> can be prevented with MAC. Feistel cipher: each block is 2w bits long, round funtion uses K1 derived from K. block = LE0 || RE0 => LE1 = RE0 and RE1 = F(K1, RE0) xor LE0 => new block = LE1 || RE1. DES (Data encryption standard): 64-bit blocks and 56-bit keys. DES makes use of 16 rounds of Feistel encryption and makes use of left shift to make "unique keys" out of the original. Tripple DES: Three stages, making use of 2 or 3 keys. 2-key variant reuses the first key in third stage, and 3-key variant makes use of 3 unique keys. E -> D -> E (Encryption), D -> E -> D (Decryption). AES: 128-bit blocks, 128, 192, 256 keys. AES general structure: Ptext = 16 bytes -> 4x4 matrix and M bytes 4x4 byte matrix -> initial transformation -> transformations (key size = 16 bytes => 10 rounds, = 24 bytes => 12 rounds, = 32 bytes => 14 rounds) -> Every round 4 transformations except round N = 3 transformations. Key format: 4x4 matrix -> w0 w1 ... wN where N = 43, 52, 60 depending on key size. Every word is 32-bit. Round x = substitute bytes -> shift rows -> Mix columns -> add round key. And key is w[x * 4, x * 4 + 3] so for round 1 => w[4, 7]. Round N = substitute bytes -> Shift rows -> Add round key. Decryption: add round key <- w[40, 43] -> Round x: inverse shift row -> inverse sub bytes -> add round key <- w[40 – x * 4, 43 – x * 4] -> inverse mix cols. Round N: inverse shift rows -> inverse sub bytes -> add round key. Fields: Fields with an inf number of elements or finite fields -> GF(p) finite fields with p elements or GF(p^n0 finite fields with p^n. GF = galois field. GF(p) for a given prime p => Zp = {0, 1, ..., p-1} if numbers are outside this scope, then number mod p = new number. AES = GF(2^8) = x^8 + x^4 + x^3 + x + 1. Addition in GF(2^8) = bit-per-bit XOR.

Multiplication in GF(2^8): polynomial multiplication modulo m(x), so in case result is > GF(2^8) => c(x) mod x^8 + x^4 + x^3 + x + 1. c(x) mod m(x) can be found with c(x) / m(x). Remainder => solution. Multiplicative Inverse in GF(2^8): Extended euclidian: Init: r_{-1} = m(x), r0 = a(x), w_{-1} = 0, w0 = , Recursive algo: ri(x) = r_{i-2}(x) mod r_{i-1}(x), qi = r_{i-2}(x)/r_{i-1}(x), wi(x) = w_{i-2}(x) - qi(x)w_{i-1}(x) Stop condition: if ri(x) = 1 => wi(x) = a^{-1}(x). In case wi(x) not in GF(2^8) => wi(x) mod m(x). AES details: Step 1: SubBytes: forward substitution byte transformation = table look up. S_{i, j}: first 4 bits = x (which column), last 4 bits = y (which row) -> mapped on S'_{i,j}. S box = 16 x 16 matrix and contains a permutation of all possible 256 8-bit values. S-box van be calculated: byte at row y and column x = yx => inverse in GF(2^8) -> convert to bit -> {b7, ..., b0}. bi' = bi xor b_{i+4 mod 8} xor b_{i+5 mod 8} xor b_{i+6 mod 8} xor b_{i+7 mod 8} xor ci and c = {c7, ..., c0} = 01100011. -> covert bit column vector b' -> byte = S(yx). Step 2: Shift rows: First row: not altered, Second row: left shift and first element to the right end, Third row: double left shift, last row: right shift. Step 3: MixColumns: s'_{0, j} = (2 * s_{0, j}) xor (3 * s_{1,j}) xor s_{2,j} xor s_{3,j}, s'_{1, j} = (2 * s_{1, j}) xor (3 * s_{2,j}) xor s_{3,j} xor s_{0,j}, s'_{2, j} = (2 * s_{2, j}) xor (3 * s_{3,j}) xor s_{0,j} xor s_{1,j}, s'_{3, j} = (2 * s_{3, j}) xor (3 * s_{0,j}) xor s_{1,j} xor s_{2,j}. Step 4: AddRoundKey: bitwise Xor 128-bit input state and 128-bit round key. 4x4 matrix xor 4x4 matrix. AES key expansion: w = B0 B1 B2 B3 -> B1 B2 B3 B0 -> Substitution using S-box table, So SubByte on Bi -> B'1 B'2 B'3 B'0 -> result xor with Rcon = RC[j] 0 0 0 and RC[1] = 1, RC[j] = 2 * RC[j-1] and multiplication defined in the field of GF(2^8). Electronic code block (ECB): plaintext split in N parts and encrypt with key K. Cipher Block Chaining (CBC): P1 xor Init vector (IV) -> encrypt with key K -> C1 = new IV, P2 xor new IV (C1) -> encrypt with key K -> C2, ..., PN xor C_{N-1} -> encrypt with key K -> CN. C = C1 || .. || CN. Cipher Feedback (CFB): stream cipher mode that operates on small blocks of s bits => IV -> Encrypt with key K -> select s bits = t1 and discard b-s bits -> P1 of size s xor t1 -> C1 -> feedback to create I2 => I_j = LSB_{b-s}(I_{j-1} || C_{j-1} and Cj = MSB_s(E(K, I_j)) xor Pj. Output Feedback (OFB): Nonce -> Encrypt with key K = O1 -> use as nonce for C2 and P1 xor O1 = C1, repeat till CN. Counter (CTR): Counter as nonce => Counter 1 -> Encrypt with key K = O1 -> P1 xor O1 -> C1. XEX-based tweaked-codebook mode with ciphertext stealing: Tweak T -> H(T) xor P -> C = E(K, H(T) xor P) xor H(T). XTS-AES: K = K1 || K2 in total 256 or 512 bits, i = 128 bit tweak, alpha = primive of GF(2^128) (GF(2^128) = x^128 + x^7 + x^2 + x + 1), a = multiplied by itself j times, j is the jthe block => E(K2, i) -> Modular multiplication of two polynomials between alpha and E(K2, i) = T-> E(K1, T xor P) xor T = C. **PART 3:** Security requirements: Variable input size = input data, Fixed output size = Output of fixed size, Efficiency = H(x) is easy to compute, Preimage resistant = given h, it is hard to find y: H(y) = h, second preimage resistant = given x, it is hard to find y: y not equal to x and H(y) = H(x), Collision resistant = it is hard to find (x,y): H(x) = H(y). SHA-512: pad message so m.length = 896 mod 1024, single 1 bit followed by 0's -> append length block of 128 bits, unsigned 128-bit integer, length of original message -> init buffer, 8 64-bit bit registers. Process message in blocks of 1024 bit -> F function -> H_{i-1} xor F_{output}. F = 80 rounds of processing making use of buffer. 😊

Length extension attack: using H(m1) and m1.length -> H(m1 || m2), this is because we use H_N = H(K || M || P) to continue our Hasing -> H'_N = H(K || M || P || M' || P'). SHA-3: sponge function with parameters: block size r and capacity c = 1600 bits. M || P = k x r bits, M = n bits -> sponge function -> r bit blocks Z_{j-1} = I + extra padding bits. Capacity reduces vulnerability to the length extension attack. MAC = message authentication code: Fixed-length value resulting from message and secret key serves as authenticator. M || H(M || S) = O => compare H(M || S) and O to check if message is not altered. Encryption as auth: yes but in case we use encryption method where content is malleable -> then no. MAC: cryptographic checksum based on a secret key. Does not need to be reversible (compared to encr). Security Req: 1. observes M and Mac(K, M) -> Infeasible to construct M' -> Mac(K, M) = Mac(K, M'). 2. two random M and M' => probability Mac(K, M) = Mac(K, M') is 2^{-n}. 3. if M' is a known transformation of M => P(Mac(K, M) = Mac(K, M')) = 2{-n}. CMAC: M1->Encr with key K (truncated k bits)->output xor M2 -> Encr with key K -> ... -> xor with Mn and output of M_{n-1} encryption and K1 -> MSB(mac with length of T) = T (left most bits). Pad incase message is not a multiple of b. HMAC: hash function based mac. Advantages: 1. faster, more support. But we need to fix that we don't use a secret key, "black box interpretation". Authenticated Encryption: Protects confidentiality and authenticity. 1. H -> E 2. A -> E 3. E -> A 4. E + A. CCM (CTR with CBC auth code): variation of E + A, 4 inputs: secret key K, plaintext P, associated data A for mac, unique nonce N. Mac = CMAC(K, N || A || P), CTR(K, P) || (MSB(E(K, Ctr0))_Tlen xor Mac). GCM (Galois / counter mode): variation of E->A. Efficient. Uses variant of CTR that includes mac. **Part 4:** RSA: p, q prime numbers, n = p * q, e: with gcd(phi(n), e) = 1 and 1 < e < phi(n) and d = e^{-1} mod phi(n) => C = M^e mod n and M = C^d mod n. PU = {e, n} and PR = {d, n}. Phi(n) = (p-1)(q-1). Efficient exponentiation: f = 1 for i = k to 0 do: f = (f * f) mod n; if bi == 1: f = (f * a) mod n; return f; Elgamel: Ingredients: q prime number, q primitive root alpha < q. Key gen: random int Xa such that 1 < Xa < q – 1. => Ya = alpha^Xa mod q public key. Encr: for M, k such that 1<= M <= q-1 => one time use key K = Ya^k mod q => C1 = alpha^k mod q and C2 = (K * M) mod q. Decr: K = C1^Xa mod q and M = (C2 * K^{-1}) mod q. Elliptic curve: Xr = (lambda^2 – Xp – Xq0 mod p and Yr = (lambda(Xp – Xr) - Yp) mod p and lambda = (Yq – Yp)/(Xq-Xp) if P not equal to Q else (3Xp^2 + a)/2Yp mod p. Digital signatures: Signing: M -> crypto hash function -> h -> signature gen algo with signers private key -> M || S. Sig verification: M || S -> M -> crypto hash function -> h -> sig veri function + sig public key + S. Must verify: author and time, authenticate the contents, verfiiable by third parties. DSA (digital signature algo): M -> H(M) -> sign with global pub key and sender private key = O -> M || O (O = s || r) -> r == verifier(s, H(M), PUg, PUa). DSA key gen: PUg: p prime number of length L (with L a multiple of 64 and at least 512), prime divisor of (p – 1) of N bits, h an integer between 1 and (p – 1), commonly h = 2 is used, g = h^{(p-1/q)} mod q and g > 1. private key: x: 0 < x < q and pub key: y = g^x mod p. RSA PSS: sign: M -> message encoding with salt -> MaskedDB || H || bc -> s = em^d mod n -> s, veri: s -> em = s^ e mod n -> maskedDB || H || bc -> veri with message M. **PART 5:** Symmetric key distribution: 1. physical delivery 2. third party physical 3. new key based on existing shared key 4. Encrypted connection with third party and exchange through encrypted link.

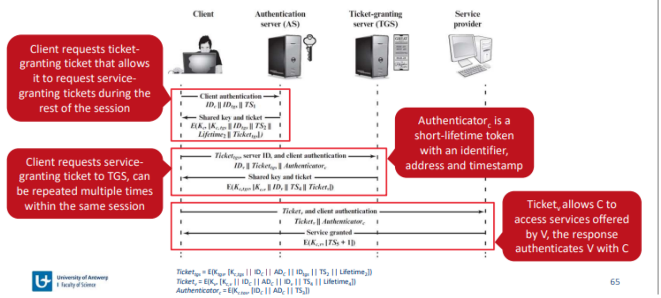
Asymmetric private key distribution: symmetric key distr with asym encr, this is vulnerable to man in the middle attack.Diffie-Hellman key exchange: q prime number, alpha primitive root of q, Xa < q and Xb < q, pub keys; Ya = alpha^Xa mod q and Yb = alpha^Xb mod q, shared key: K = Yb^Xa mod q = Ya^Xb mod q. Still susceptible to man in the middle attack. Public key distribution: 1. public announcement 2. public directory 3. public key authority 4. public key certificates. Public key authority: 1. A -> PKA: request || T1 2. PKA -> A: E(PRauth, (PUb || Request || T1)) 3. A -> B: E(PUb, (IDa || N1)) 4. B -> PKA: Request || T2 5. PKA -> B: E(PRauth, (PUa || Request || T2)) 6. B -> A: E(PUa, (N1 || N2)) 7. A -> B: E(PUb, N2). Certificates: Secure key exchange without an auth. Reqr: 1. pub key and name deterministic 2. verifiable it came from CA 3. Only CA can create/modify certificate 4. validity/time deterministic. Step 1: 1. A->CA: PUa 2. CA -> A: CA = E(PRauth, (T1 || IDa || PUa)) 3. B -> CA: PUb 4. E(PRauth, (T2 || IDb || PUb)) Step 2: 1. A -> B: Ca 2. B->A: Cb. X.509 uses hash based signatures:



Unsigned certificate: contains user ID, user's public key
Bob's ID information
Bob's public key
CA information
H
E
H
D
Recipient can verify signature by comparing hash code values
Generate hash code of unsigned certificate
Signed certificate
Encrypt hash code with CA's private key to form signature
Decrypt signature with CA's public key to recover hash code
Create signed digital certificate
Use certificate to verify Bob's public key

Version: Differentiates among successive versions of the certificate format; Serial number: An integer value unique within the issuing CA. Signature algorithm identifier: The algorithm used to sign the certificate together. Issuer name. Period of validity: Consists of two dates: the first and last on which the certificate is valid. Subject name: The name of the user to whom this certificate refers. Subject's public-key information: The public key of the subject, plus an identifier of the algorithm for which this key is to be used. Issuer unique identifier: An optional-bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities. Subject unique identifier: An optional-bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities. Extensions. Signature: Covers all of the other fields of the certificate. One component of this field is the digital signature applied to the other fields of the certificate. This field includes the signature algorithm identifier. Certificate organisation: Y<<X>> = Certificate of user X issued by CA Y. Certificate revocation: revoke a certificate that is not expired. Each CA maintains certificate revocation list, not expired but revoked certificates. Public Key Infrastructure: set of hardware, software, people, policies and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography. Objective: Enable secure, convenient, and efficient acquisition of public keys.

User management: Authentication = verify identity of a system entity -> Authorization = verify entity's right to perform specific actions. Means of auth: Knows, Has, Is. Mutual auth: mutually agree on each other's identity and securely exchanging session keys. Key distribution center: 1. A->KDC: IDa || IDb || N1 2. E(Ka, (Ks || IDa || IDb || N1)) || E(Kb, (Ks || IDa)) 3. A->B: E(Kb, (Ks || IDa)) 4. B->A: E(Ks, N2) 5. A->B: E(Ks, f(N2)). Last two steps prevent replay attacks, if old session key is intercepted -> replay can still be performed. Add timestamp on B based on local clock: 1. A->B: IDa || Na 2. B->KDC: IDb || Nb || E(Kb, (IDa || Na || Tb)) 3. KDC->A: E(Ka, (IDb || Na || Ks || Tb)) || E(Kb, (IDa || Ks || Tb)) || Nb 4. A->B: E(Kb, (IDa || Ks || Tb)) || E(Ks,Nb). New session within time scope, no need of KDC. Kerberos: Allow servers in a distributed environment to restrict access to authorized users and authenticate requests for service, where users can make use of different (untrusted) workstations. 😊



Client
Authentication server (AS)
Ticket-granting server (TGS)
Service provider
Client requests ticket-granting ticket that allows it to request service-granting tickets during the rest of the session
Client authentication
$ID_c \| ID_{tgs} \| TS_1$
Shared key and ticket
$E(K_c, [K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$
Client requests service-granting ticket to TGS, can be repeated multiple times within the same session
$Ticket_{tgs}$, server ID, and client authentication
$ID_v \| Ticket_{tgs} \| Authenticator_c$
Shared key and ticket
$E(K_{c,tgs}, [K_{c,v} \| ID_v \| TS_4 \| Ticket_v])$
Authenticator$_c$ is a short-lifetime token with an identifier, address and timestamp
$Ticket_v$ and client authentication
$Ticket_v \| Authenticator_c$
Service granted
$E(K_{c,v}, [TS_5 + 1])$
Ticket$_v$ allows C to access services offered by V, the response authenticates V with C
$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} || ID_C || AD_C || ID_{tgs} || TS_2 || Lifetime_2])$
$Ticket_v = E(K_v, [K_{c,v} || ID_C || AD_C || ID_v || TS_4 || Lifetime_4])$
$Authenticator_c = E(K_{c,tgs}, [ID_C || AD_C || TS_3])$
University of Antwerp | Faculty of Science
65

PART 6: Wired equivalent protection: passive attack to discover key with RC4. Wi-fi protected access 2: Robust Security Network = {Access control, Authentication, privacy with message integrity}. WPA2 phases: 1. Discovery: An AP uses messages called Beacons and Probe Responses to advertise its security policy. The STA uses these to identify an AP for a WLAN with which it wishes to communicate. The STA associates with the AP, which it uses to select the cipher suite and authentication mechanism when the Beacons and Probe Responses present a choice. 2. Authentication: The STA and AS prove their identities to each other. The AP blocks no authentication traffic between the STA and AS until the authentication transaction is successful. The AP does not participate in the authentication transaction other than forwarding traffic between the STA and AS. 3. Key generation and distribution: The AP and the STA perform several operations that cause cryptographic keys to be generated and placed on the AP and the STA. Frames are exchanged between the AP and STA only. 4. Protected data transfer: Frames are exchanged between the STA and the end station through the AP. As denoted by the shading and the encryption module icon, secure data transfer occurs between the STA and the AP only; security is not provided end-to-end. 5. Connection termination: During this phase, the secure connection is torn down, and the connection is restored to the original state. Extensible authentication protocol (EAP): EAP-TLS = Unencrypted Transport layer security handshake that uses certificates. EAP-TTLS (tunneled tls) = server uses EAP-TLS and client can use any EAP authentication method, EAP-GPSK (generalized pre shared key) = auth and ses keys are derived from pre shared private key, EAP-IKEv2 (internet key exchange protocol): IKE is used for mutual auth.

Key management: master-key (PSK or MSK) -> Pairwise master key -> Pairwise transient key -> key confirmation key, key encryption key, temporal key. Group master key -> group temporal key. Protected data transfer: Temporal Key Integrity Protocol = TKIP: Hardware compatible with WEP => software changes only, Integrity 64-bit MIC, confidentiality: MPDU+MIC encr with RC4. Counter mode-CBC mac protocol (ccmp): dedicated hardware, integrity, confidentiality: CTR block cipher with AES-128. Key gen: TK = 256 bits => TKIP = two michael keys (64-bit x 2) | RC4 key (64 or 128), CCMP = AES key (used for both) 128 bits. WPA3: Extends the dragonfly handshake (Diffie Hellman) to make pairwise PMK from PSK. IPsec (Ip layer security): Transport mode (Only IP packet payload encrypted) - Tunnel mode (Entire IP packet is encrypted, authenticated, encapsulated into new IP packet). Anti-replay function: we keep a fixed window size, mark if packet is valid and received. Packets with sequence number <= N-W are considered replays. Advance window to the right if valid packet is received. Internet key exchange: Determination and distribution of secret keys for confidentiality and integrity. Transport layer security (TLS): Two layers of protocols: TLS record protocol, {Handshake, change cipher spec, alert, heartbeat} protocol. TLS record protocol: application data -> fragment -> optional compression -> add mac -> encrypt -> append ssl record header. Handshake protocol: 1. establish capabilities: hello messages to exchange nonce, key material, supported versions, supported crypto algo 2. server auth: depending on key exchange method, server sends certificate, public key and/or client certificate request. 3. client auth: client sends certificate if requested, and verification (optional) and its own pub key 4. finish: part of cipher spec protocol, new algo and new keys. Heartbleed: if pl array is smaller than what payload size makes believe, old data in memory allotted to bp array is not
Overwritten (memcpy(bp,pl,payloadsize). POODLE: trick server to downgrade to SSL 3.0 and use padding oracle man in the middle attack. Knowing the padding format and it uses cbc, we can find plaintext bye by byte. SSH: Insecure connection -> secure connection using SSH tunnel. S/MIME = secure multipurpose internet mail extensions: extra algo to secure MIME. M -> sign with senders private key -> M || sig -> Encrypt with one time key -> encrypt one time key with receivers pub key -> E(PUr, K) || E(K, M || sig). PGP compression: compression after: 1. less storage 2. zip is non-deterministic, compression before: stronger crypto security. Firewall types: packet filtering, stateful inspection, application proxy, circuit level proxy. DDos attacks: Syn flood attack (internal recourse attack), ICMP attack (data transmission attack). Top 10 web appli security risks: 1. broken access control 2. crypto failures 3. injection 4. insecure design 5. security misconfiguration 6. vulnerable and outdated components 7. identification and auth fails 8. software and data integrity fails 9. logging and monitoring fails 10. server side request forgery. XML External Entity (XXE): Poorly configured XML parsers are vulnerable, exploits the parser desire to blindly execute process external DTD entities. Cross-site scripting (XSS): injecting user defined scripts in trusted websites without validating the script. Stored = permanently stored on target servers. Reflected: is reflected of the server, delivered to the victim using some other way. (The end, good luck and yeah) - Made by Jason Liu