CSE 490K - Lecture 7

Physical Security and Computer Security and Public Key Cryptography

Tadayoshi Kohno

Some slides derived from Vitaly Shmatikov's

Security Mindset

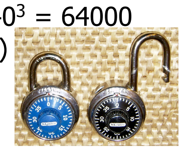
- The "security mindset" is > 1/2 of computer security; maybe much more than 1/2
  - Informal, heuristic, but seems to be true
  - Technical tools help, but are ineffective if used improperly
  - Need to think like the "bad guy"
    - But don't be bad (recall the Ethics Form)!
    - Every single line of code may be the target of an adversary
    - Adversaries may be foreign nations
    - Adversaries only need to find one way to "win"
    - ...
  - Goal: Think like the "bad guy" -- at least spot problems, even if don't know how to fix

Last Class

- Relate physical security to computer security
  - Locks, safes, etc
- Why?
  - More similar than you might think!!
  - Lots to learn:
    - Computer security issues are often very abstract; hard to relate to
    - But physical security issues are often easier to understand
  - Hypothesis:
    - Thinking about the "physical world" in new (security) ways will help you further develop the "security mindset"
    - You can then apply this mindset to computer systems, ...
  - Plus, communities can learn from each other

Master Combination Locks

- 40 positions, 3 numbers in "key:" 40^3 = 64000 possible combinations (theoretically)
- But really only 2560 possibilities



- And using some tricks, can figure out key:
  - Average of 32 tries (old locks)
    - Easy to "feel" last number in key; then only 64 remaining possibilities
  - Average of 128 tries (new locks)
- Another example of a "shortcut attack" against the key

Some "Big Picture" Issues

- Don't rely on "security through obscurity"
  - Easy to learn how locks work
    - Insiders
    - Tinkerers
  - Easy to learn how software works
    - Insiders
    - Tinkerers
    - Examples: DRM, reverse engineering software patches
- Have an open, peer-reviewed (or at least outside expert-reviewed) design

Some "Big Picture" Issues

- Usability is a major challenge
  - Locks:
    - If locks are too complicated, people may not use them
      - But then locks don't provide any security
    - See Blaze's "safecracking" paper for an example - **class 1** safes are more secure, but have awkward security mechanisms
  - Computers:
    - If security mechanisms are too difficult, people won't use them
    - Example: Personal firewall or antivirus warnings
- Make "secure option" the "default" or "option of least resistance"

Some "Big Picture" Issues

- Many potential ways to compromise security
  - Physical security
    - Attack locks
    - Attack the door itself
    - Attack windows
    - Hide in bushes
  - Computer security
    - Attack the cryptography (if done poorly)
    - Attack the configuration
    - Attack the implementation
    - Attack the user
- "Security only as strong as the weakest link"
- Systems are <u>complex</u>

Some "Big Picture" Issues

- Not all systems require the same level of security
  - Locks
    - Weak locks may be OK to protect you gym cloths
    - But may want stronger locks to protect the contents of your bank's safes
  - Computer security
    - Different assets, adversaries, protection mechanisms
- "Security is risk management"

Some "Big Picture" Issues

- Defense in depth
  - Physical world
    - Layers of locks in bank
    - Layers of protection mechanisms around jails
    - Castles: Moats, walls, arrows, ...
  - Digital world
    - Same concepts apply
- Deterrents
  - Physical world
    - Video cameras
    - ADT (home security alarm system)
  - Digital world
    - Digital forensics methods

Some "Big Picture" Issues

- Packaging (sometimes called "snake oil")
  - Physical world
    - May **look** secure, but may be easy to circumvent
  - Digital world
    - May **appear** secure, but may actually be very insecure
- How is a user supposed to figure out whether something is secure?

Some "Big Picture" Issues

- Issues at all phases of development lifecycle
  - Physical world
    - Requirements: Master keys (whether to have or not)
    - Design: Master keys (design choices, e.g., master pin depths)
    - Implementation: Lock picking
  - Digital world
    - Same issues apply
- Better to address security issues as early in the lifecycle as possible

Some "Big Picture" Issues

- Denial of service
  - Locks
    - Chewing gum
    - Super glue
    - Break a key
  - Computers
    - Crash computer, consume resources
- Accidents
  - Locks
    - Keys on both sides (fire hazard)
  - Computers
    - Encrypted filesystem (forget key)
    - ...

Some "Big Picture" Issues

- Many different adversaries
  - Insiders
  - Ex-insiders (past employees, with copies of keys)
  - Pranksters
  - Outsiders
  - …

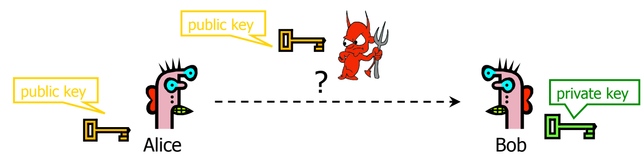
Some "Big Picture" Issues

- Arms race
  - Physical world
    - New lock designs, better safes
  - Digital world
    - New cryptography
    - New software development practices
    - Software updates

Some "Big Picture" Issues

- Big difference: Connectedness
  - Physical world
    - Not very connected
    - (Yes, some exceptions, e.g., postal system or air travel)
  - Digital world
    - Everyone is everyone else's "neighbor"
    - Plus quite a bit of anonymity

Basic Problem



Given: Everybody knows Bob's public key
Only Bob knows the corresponding private key

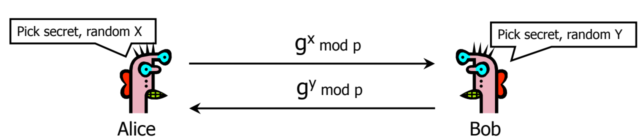
Goals: 1. Alice wants to send a secret message to Bob
2. Bob wants to authenticate himself

Applications of Public-Key Crypto

- Encryption for confidentiality
  - Anyone can encrypt a message
    - With symmetric crypto, must know secret key to encrypt
  - Only someone who knows private key can decrypt
  - Key management is simpler (maybe)
    - Secret is stored only at one site: good for open environments
- Digital signatures for authentication
  - Can "sign" a message with your private key
- Session key establishment
  - Exchange messages to create a secret session key
  - Then switch to symmetric cryptography (why?)

Diffie-Hellman Protocol (1976)

- Alice and Bob never met and share no secrets
- Public info: p and g
  - p is a large prime number, g is a generator of Z_p^*
    - $Z_p^*=\{1, 2 \dots p-1\}$; $\forall a \in Z_p^*$ $\exists i$ such that $a=g^i \bmod p$
    - Modular arithmetic: numbers "wrap around" after they reach p



Compute $k=(g^y)^x=g^{xy} \bmod p$ Compute $k=(g^x)^y=g^{xy} \bmod p$

Why Is Diffie-Hellman Secure?

- Discrete Logarithm (DL) problem:
  given $g^x \bmod p$, it's hard to extract x
  - There is no known efficient algorithm for doing this
  - This is not enough for Diffie-Hellman to be secure!
- Computational Diffie-Hellman (CDH) problem:
  given g^x and g^y, it's hard to compute $g^{xy} \bmod p$
  - … unless you know x or y, in which case it's easy
- Decisional Diffie-Hellman (DDH) problem:
  given g^x and g^y, it's hard to tell the difference between $g^{xy} \bmod p$ and $g^r \bmod p$ where r is random

Properties of Diffie-Hellman

- Assuming DDH problem is hard, Diffie-Hellman protocol is a secure key establishment protocol against passive attackers
  - Eavesdropper can't tell the difference between established key and a random value
  - Can use new key for symmetric cryptography
    - Approx. 1000 times faster than modular exponentiation
- Diffie-Hellman protocol (by itself) does not provide authentication

Requirements for Public-Key Crypto

- Key generation: computationally easy to generate a pair (public key PK, private key SK)
  - Computationally infeasible to determine private key SK given only public key PK
- Encryption: given plaintext M and public key PK, easy to compute ciphertext $C=E_{PK}(M)$
- Decryption: given ciphertext $C=E_{PK}(M)$ and private key SK, easy to compute plaintext M
  - Infeasible to compute M from C without SK
  - Even infeasible to learn partial information about M
  - Trapdoor function: Decrypt(SK,Encrypt(PK,M))=M

Some Number Theory Facts

- Euler totient function $\varphi(n)$ where $n \geq 1$ is the number of integers in the [1,n] interval that are relatively prime to n
  - Two numbers are relatively prime if their greatest common divisor (gcd) is 1
- Euler's theorem:
  if $a \in Z_n^*$, then $a^{\varphi(n)}=1 \bmod n$
- Special case: Fermat's Little Theorem
  if p is prime and gcd(a,p)=1, then $a^{p-1}=1 \bmod p$

RSA Cryptosystem

[Rivest, Shamir, Adleman 1977]

- Key generation:
  - Generate large primes p, q
    - Say, 1024 bits each (need primality testing, too)
  - Compute $n=pq$ and $\varphi(n)=(p-1)(q-1)$
  - Choose small e, relatively prime to $\varphi(n)$
    - Typically, $e=3$ or $e=2^{16}+1=65537$ (why?)
  - Compute unique d such that $ed = 1 \bmod \varphi(n)$
  - Public key = (e,n); private key = d
- Encryption of m: $c = m^e \bmod n$
  - Modular exponentiation by repeated squaring
- Decryption of c: $c^d \bmod n = (m^e)^d \bmod n = m$

Why RSA Decryption Works

- $e \cdot d=1 \bmod \varphi(n)$
- Thus $e \cdot d=1+k \cdot \varphi(n)=1+k(p-1)(q-1)$ for some k
- Let m be any integer in Z_n
- If gcd(m,p)=1, then $m^{ed}=m \bmod p$
  - By Fermat's Little Theorem, $m^{p-1}=1 \bmod p$
  - Raise both sides to the power k(q-1) and multiply by m
  - $m^{1+k(p-1)(q-1)}=m \bmod p$, thus $m^{ed}=m \bmod p$
  - By the same argument, $m^{ed}=m \bmod q$
- Since p and q are distinct primes and $p \cdot q=n$,
  $m^{ed}=m \bmod n$

Why Is RSA Secure?

- RSA problem: given $n=pq$, e such that $\gcd(e,(p-1)(q-1))=1$ and c, find m such that $m^e=c \bmod n$
  - i.e., recover m from ciphertext c and public key (n,e) by taking e^{th} root of c
  - There is no known efficient algorithm for doing this
- Factoring problem: given positive integer n, find primes $p_1, \ldots, p_k$ such that $n=p_1^{e_1}p_2^{e_2}\ldots p_k^{e_k}$
- If factoring is easy, then RSA problem is easy, but there is no known reduction from factoring to RSA
  - It may be possible to break RSA without factoring n

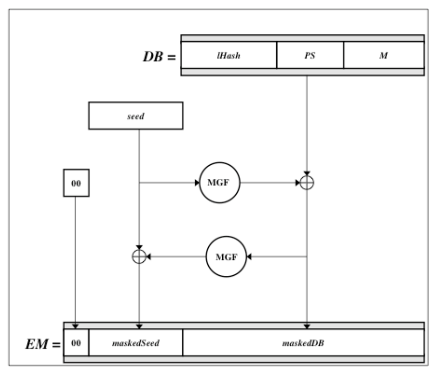
Caveats

- e =3 is a common exponent
  - If $m < n^{1/3}$, then $c = m^3 < n$ and can just take the cube root of c to recover m
    - Even problems if "pad" m in some ways [Hastad]
  - Let $c_i = m^3 \bmod n_i$ - same message is encrypted to three people
    - Adversary can compute $m^3 \bmod n_1n_2n_3$ (using CRT)
    - Then take ordinary cube root to recover m
- Don't use RSA directly

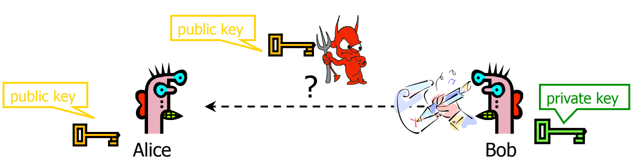
Integrity in RSA Encryption

- Plain RSA does <u>not</u> provide integrity
  - Given encryptions of m_1 and m_2, attacker can create encryption of $m_1 \cdot m_2$
    - $(m_1^e) \cdot (m_2^e) \bmod n = (m_1 \cdot m_2)^e \bmod n$
  - Attacker can convert m into m^k without decrypting
    - $(m_1^e)^k \bmod n = (m^k)^e \bmod n$
- In practice, OAEP is used: instead of encrypting M, encrypt $M \oplus G(r)$; $r \oplus H(M \oplus G(r))$
  - r is random and fresh, G and H are hash functions
  - Resulting encryption is plaintext-aware: infeasible to compute a valid encryption without knowing plaintext
    - … if hash functions are "good" and RSA problem is hard

OAEP (image from PKCS #1 v2.1)



Digital Signatures: Basic Idea



<u>Given</u>: Everybody knows Bob's public key
Only Bob knows the corresponding private key

<u>Goal</u>: Bob sends a "digitally signed" message

1. To compute a signature, must know the private key
2. To verify a signature, enough to know the public key

RSA Signatures

- Public key is (n,e), private key is d
- To sign message m: $s = m^d \bmod n$
  - Signing and decryption are the same operation in RSA
  - It's infeasible to compute s on m if you don't know d
- To verify signature s on message m: $s^e \bmod n = (m^d)^e \bmod n = m$
  - Just like encryption
  - Anyone who knows n and e (public key) can verify signatures produced with d (private key)
- In practice, also need padding & hashing (why?)

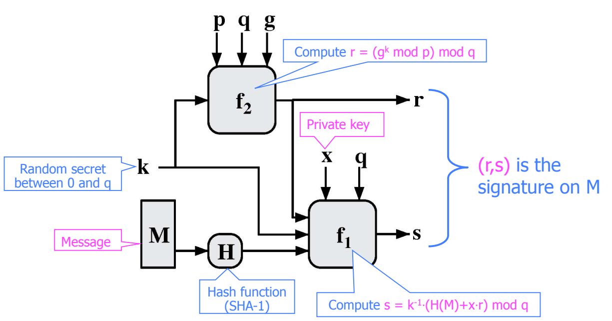
Encryption and Signatures

- Book says: Encryption and decryption are inverses.
- That's a common view
  - True for the RSA primitive
- But not one we'll take
  - To really use RSA, we need padding
  - And there are many other decryption methods

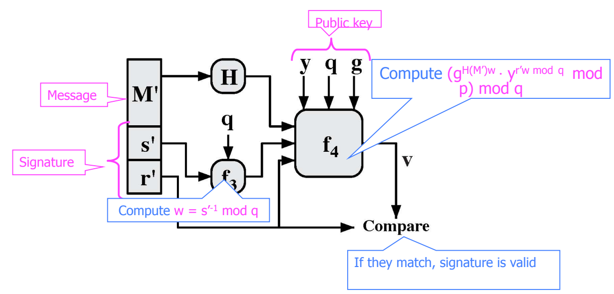
Digital Signature Standard (DSS)

- U.S. government standard (1991-94)
  - Modification of the ElGamal signature scheme (1985)
- Key generation:
  - Generate large primes p, q such that q divides p-1
    - $2^{159} < q < 2^{160}$, $2^{511+64t} < p < 2^{512+64t}$ where $0 \le t \le 8$
  - Select $h \in Z_p^*$ and compute $g = h^{(p-1)/q}$ mod p
  - Select random x such $1 \le x \le q-1$, compute $y = g^x$ mod p
- Public key: (p, q, g, $y = g^x$ mod p), private key: x
- Security of DSS requires hardness of discrete log
  - If could solve discrete logarithm problem, would extract x (private key) from g^x mod p (public key)

DSS: Signing a Message



DSS: Verifying a Signature



Why DSS Verification Works

- If (r,s) is a legitimate signature, then
  $r = (g^k \bmod p) \bmod q$; $s = k^{-1} \cdot (H(M) + x \cdot r) \bmod q$
- Thus $H(M) = -x \cdot r + k \cdot s \bmod q$
  - Multiply both sides by $w = s^{-1} \bmod q$
- $H(M) \cdot w + x \cdot r \cdot w = k \bmod q$
  - Exponentiate g to both sides
- $(g^{H(M) \cdot w + x \cdot r \cdot w} = g^k) \bmod p \bmod q$
  - In a valid signature, $g^k \bmod p \bmod q = r$, $g^x \bmod p = y$
- Verify $g^{H(M) \cdot w} \cdot y^{r \cdot w} = r \bmod p \bmod q$

Security of DSS

- Can't create a valid signature without private key
- Given a signature, hard to recover private key
- Can't change or tamper with signed message
- If the same message is signed twice, signatures are different
  - Each signature is based in part on random secret k
- Secret k must be different for each signature!
  - If k is leaked or if two messages re-use the same k, attacker can recover secret key x and forge any signature from then on

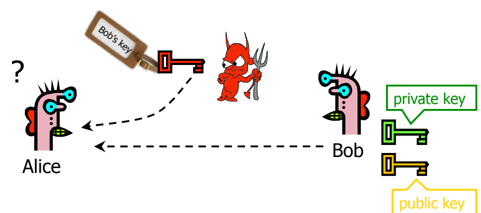
Advantages of Public-Key Crypto

- Confidentiality without shared secrets
  - Very useful in open environments
  - No "chicken-and-egg" key establishment problem
    - With symmetric crypto, two parties must share a secret before they can exchange secret messages
    - Caveats to come
- Authentication without shared secrets
  - Use digital signatures to prove the origin of messages
- Reduce protection of information to protection of authenticity of public keys
  - No need to keep public keys secret, but must be sure that Alice's public key is <u>really</u> her true public key

Disadvantages of Public-Key Crypto

- Calculations are 2-3 orders of magnitude slower
  - Modular exponentiation is an expensive computation
  - Typical usage: use public-key cryptography to establish a shared secret, then switch to symmetric crypto
    - We'll see this in IPSec and SSL
- Keys are longer
  - 1024 bits (RSA) rather than 128 bits (AES)
- Relies on unproven number-theoretic assumptions
  - What if factoring is easy?
    - Factoring is <u>believed</u> to be neither P, nor NP-complete
  - (Of course, symmetric crypto also rests on unproven assumptions)

Authenticity of Public Keys

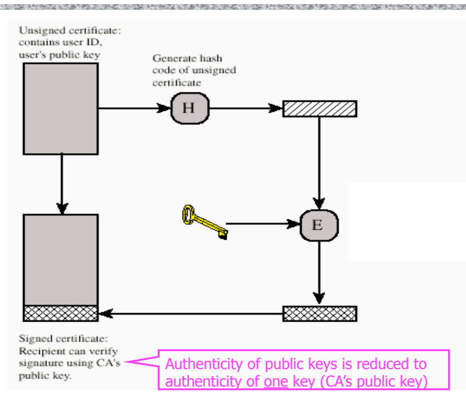


<u>Problem</u>: How does Alice know that the public key she received is really Bob's public key?

Distribution of Public Keys

- Public announcement or public directory
  - Risks: forgery and tampering
- Public-key certificate
  - Signed statement specifying the key and identity
    - sig_{Alice}("Bob", PK_B)
- Common approach: certificate authority (CA)
  - Single agency responsible for certifying public keys
  - After generating a private/public key pair, user proves his identity and knowledge of the private key to obtain CA's certificate for the public key (offline)
  - Every computer is <u>pre-configured</u> with CA's public key

Using Public-Key Certificates



Hierarchical Approach

- Single CA certifying every public key is impractical
- Instead, use a trusted root authority
  - For example, Verisign
  - Everybody must know the public key for verifying root authority's signatures
- Root authority signs certificates for lower-level authorities, lower-level authorities sign certificates for individual networks, and so on
  - Instead of a single certificate, use a certificate chain
    - $sig_{Verisign}$("UW", PK_{UW}), sig_{UW}("Alice", PK_A)
  - What happens if root authority is ever compromised?

Many Challenges

Spoofing URLs With Unicode

Posted by timothy **on Mon May 27, '02 09:48 PM**
from the **there-is-a-problem-with-this-certificate** dept.

Embedded Geek writes:

"Scientific American has an interesting article about how a pair of students at the Technion-Israel Institute of Technology registered "microsoft.com" with Verisign, using the Russian Cyrillic letters "c" and "o". Even though it is a completely different domain, the two display identically (the article uses the term "homograph"). The work was done for a paper in the **Communications of the ACM** (the paper itself is not online). The article characterizes attacks using this spoof as "scary, if not entirely probable," assuming that a hacker would have to first take over a page at another site. I disagree: sending out a mail message with the URL waiting to be clicked ("Bill Gates will send you ten dollars!") is just one alternate technique. While security problems with Unicode have been noted here before, this might be a new twist."

Many Challenges

Shmoo Group Finds Exploit For non-IE Browsers

Posted by Hemos **on Mon Feb 07, '05 11:30 AM**
from the **even-mozilla-is-guilty** dept.

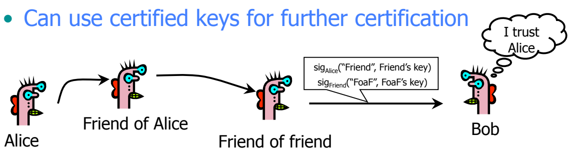
shut_up_man writes

"Saw this on Boing Boing: East coast hacker con Shmoocon ended today and they had a nasty browser exploit to show off... using International Domain Name (IDN) character support to display fake domain names in links and the address bar. Their examples use Paypal (with SSL too) and this looks very useful for phishing attacks. Interesting note that it works in every browser *except* IE (which makes this exploit a lot less dangerous in the end, I suppose)."

v The reason IE isn't vulnerable is because it doesn't natively support IDN; with the right plug-in, it too is vulnerable.

Alternative: "Web of Trust"

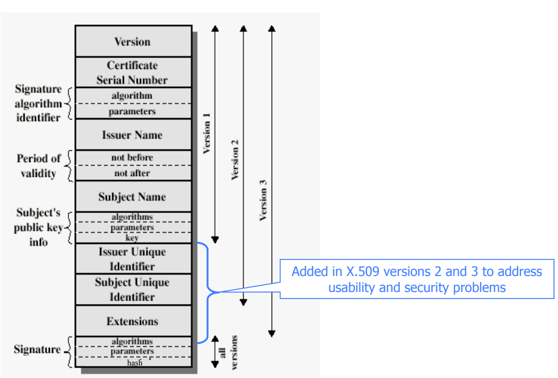
- Used in PGP (Pretty Good Privacy)
- Instead of a single root certificate authority, each person has a set of keys they "trust"
  - If public-key certificate is signed by one of the "trusted" keys, the public key contained in it will be deemed valid
- Trust can be transitive
  - Can use certified keys for further certification



X.509 Authentication Service

- Internet standard (1988-2000)
- Specifies certificate format
  - X.509 certificates are used in IPSec and SSL/TLS
- Specifies certificate directory service
  - For retrieving other users' CA-certified public keys
- Specifies a set of authentication protocols
  - For proving identity using public-key signatures
- Does not specify crypto algorithms
  - Can use it with any digital signature scheme and hash function, but hashing is required before signing

X.509 Certificate



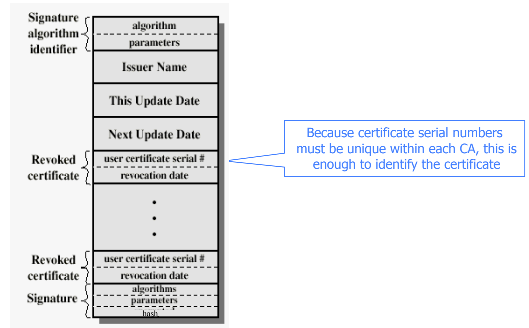
Certificate Revocation

- Revocation is very important
- Many valid reasons to revoke a certificate
  - Private key corresponding to the certified public key has been compromised
  - User stopped paying his certification fee to this CA and CA no longer wishes to certify him
  - CA's certificate has been compromised!
- Expiration is a form of revocation, too
  - Many deployed systems don't bother with revocation
  - Re-issuance of certificates is a big revenue source for certificate authorities

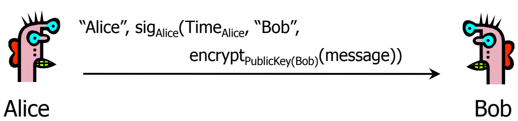
Certificate Revocation Mechanisms

- Online revocation service
  - When a certificate is presented, recipient goes to a special online service to verify whether it is still valid
    - Like a merchant dialing up the credit card processor
- Certificate revocation list (CRL)
  - CA periodically issues a signed list of revoked certificates
    - Credit card companies used to issue thick books of canceled credit card numbers
  - Can issue a "delta CRL" containing only updates
- Question: does revocation protect against forged certificates?

X.509 Certificate Revocation List

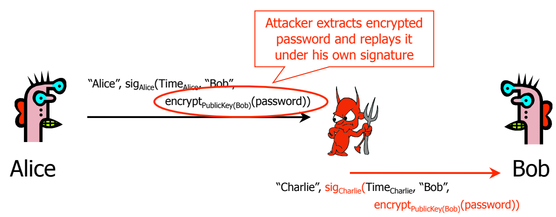


X.509 Version 1



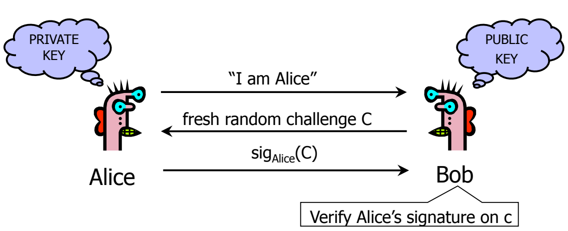
- Encrypt, then sign for authenticated encryption
  - Goal: achieve both confidentiality and authentication
  - E.g., encrypted, signed password for access control
- Does this work?

Attack on X.509 Version 1



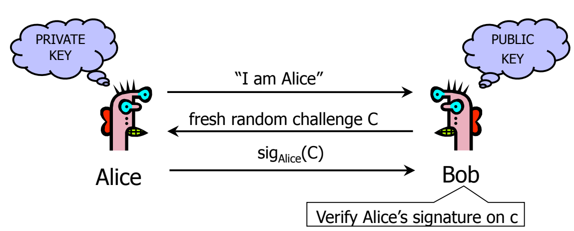
- Receiving encrypted password under signature does <u>not</u> mean that the sender actually knows the password!

Authentication with Public Keys



1. Only Alice can create a valid signature
2. Signature is on a fresh, unpredictable challenge

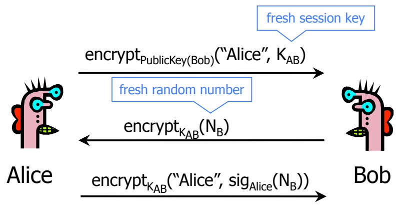
Authentication with Public Keys



1. Only Alice can create a valid signature
2. Signature is on a fresh, unpredictable challenge

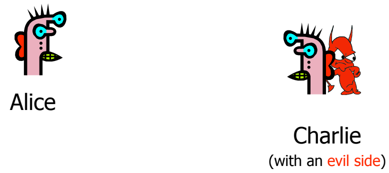
Potential problem: Alice will sign <u>anything</u>

Early Version of SSL (Simplified)



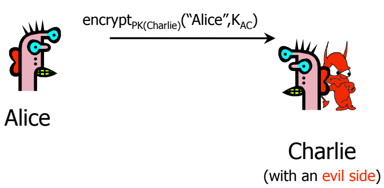
- Bob's reasoning: I must be talking to Alice because…
  - Whoever signed N_B knows Alice's private key… Only Alice knows her private key… Alice must have signed N_B… N_B is fresh and random and I sent it encrypted under K_{AB}… Alice could have learned N_B only if she knows K_{AB}… She must be the person who sent me K_{AB} in the first message...

Breaking Early SSL



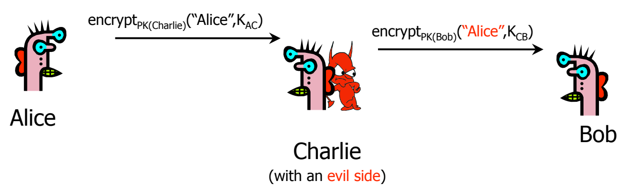
- Charlie uses his legitimate conversation with Alice to impersonate Alice to Bob
  - Information signed by Alice is not sufficiently explicit

Breaking Early SSL



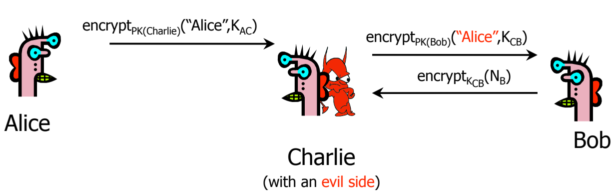
- Charlie uses his legitimate conversation with Alice to impersonate Alice to Bob
  - Information signed by Alice is not sufficiently explicit

Breaking Early SSL



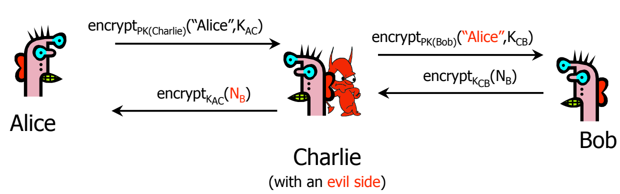
- Charlie uses his legitimate conversation with Alice to impersonate Alice to Bob
  - Information signed by Alice is not sufficiently explicit

Breaking Early SSL



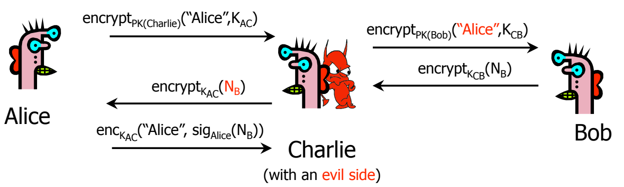
- Charlie uses his legitimate conversation with Alice to impersonate Alice to Bob
  - Information signed by Alice is not sufficiently explicit

Breaking Early SSL



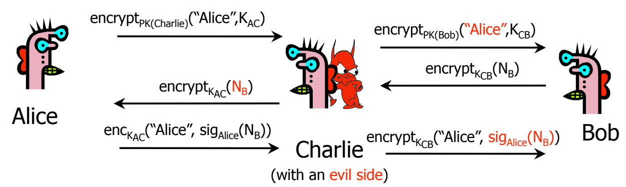
- Charlie uses his legitimate conversation with Alice to impersonate Alice to Bob
  - Information signed by Alice is not sufficiently explicit

Breaking Early SSL



- Charlie uses his legitimate conversation with Alice to impersonate Alice to Bob
  - Information signed by Alice is not sufficiently explicit

Breaking Early SSL



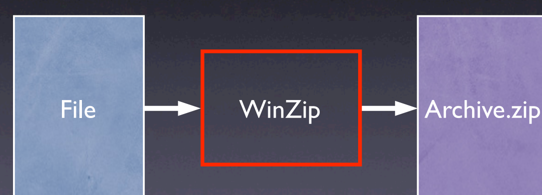
- Charlie uses his legitimate conversation with Alice to impersonate Alice to Bob
  - Information signed by Alice is not sufficiently explicit

Security Evaluation #2

- You'll be looking at WinZip's new AE-2 encryption scheme
  - Based on "Encrypt-then-MAC" (recall a few classes ago --- this is a provably secure mode)
  - But things aren't always that simple
    - Many protocols seem secure but actually have problems
  - Your job: Analyze AE-2

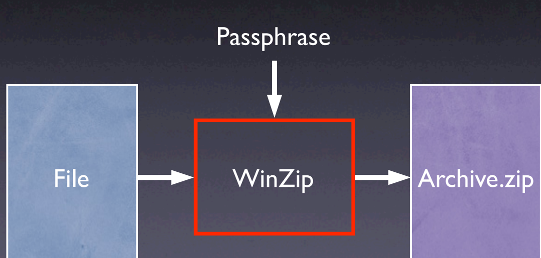
What is WinZip?

Very popular Windows compression utility. Also an Outlook email plugin. Over 160 million downloads from download.com alone [http://www.winzip.com/empopp.htm].

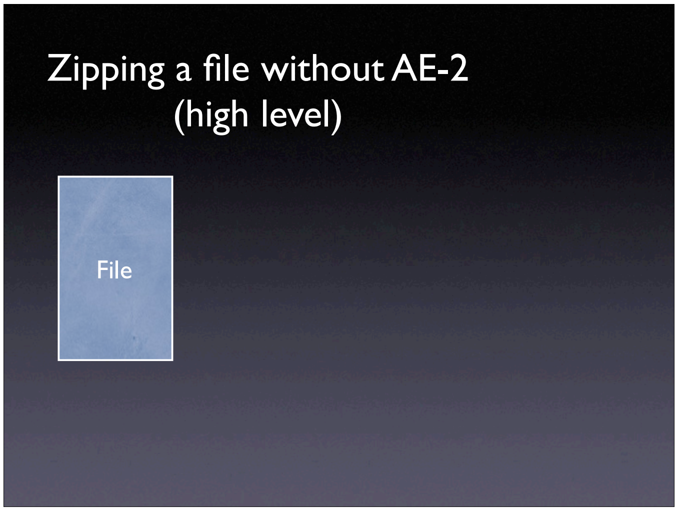


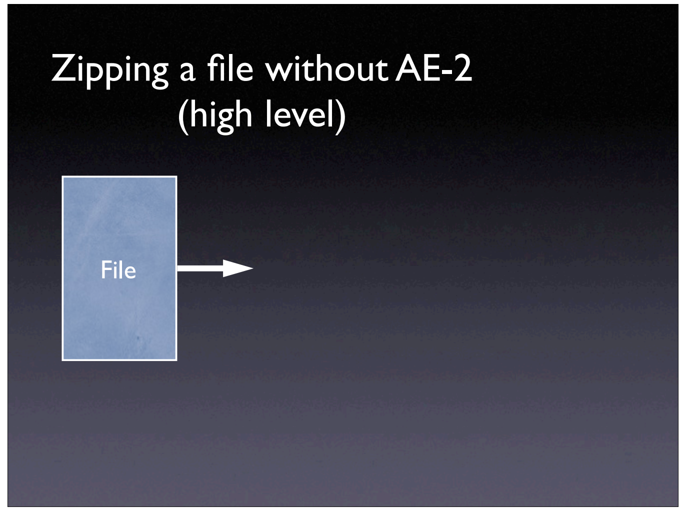
WinZip encryption

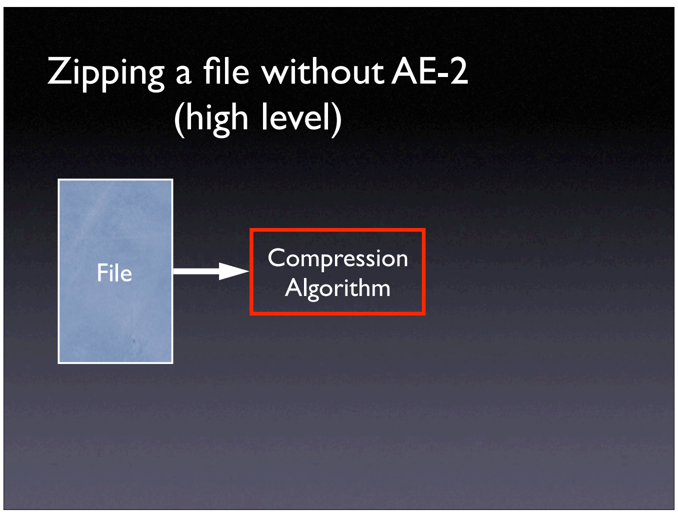
WinZip has the ability to encrypt files. Lots of history, but we'll look at the AE-2 method.

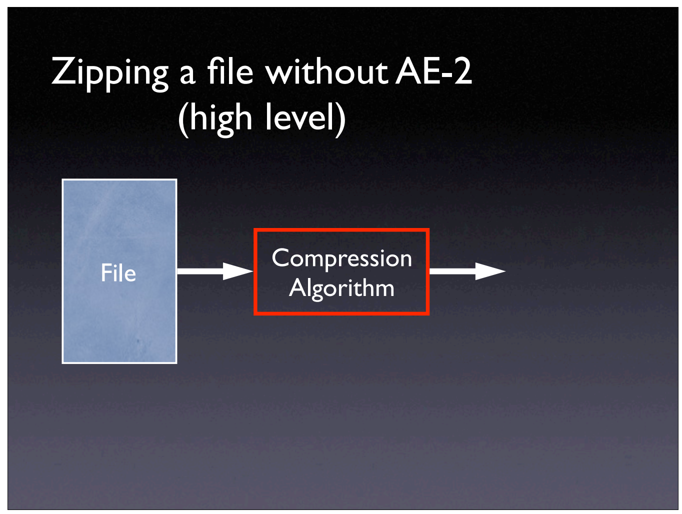


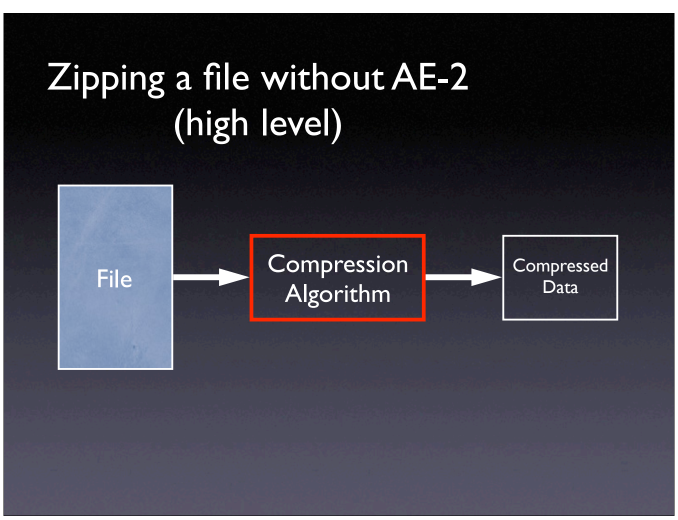
Zipping a file without AE-2 (high level)

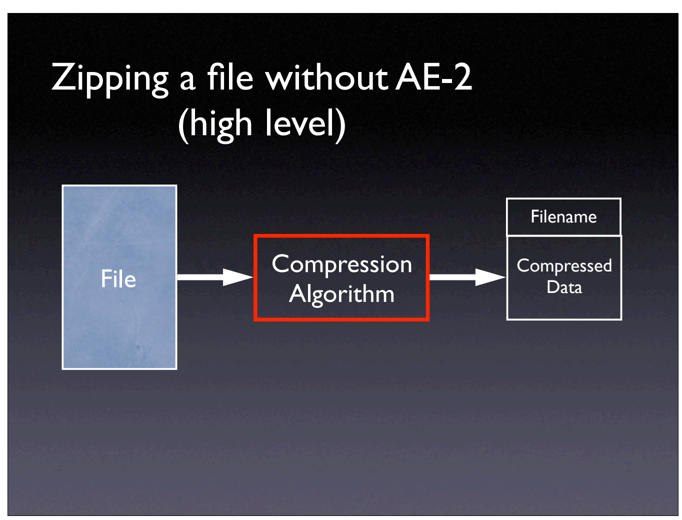

Zipping a file without AE-2
(high level)
File

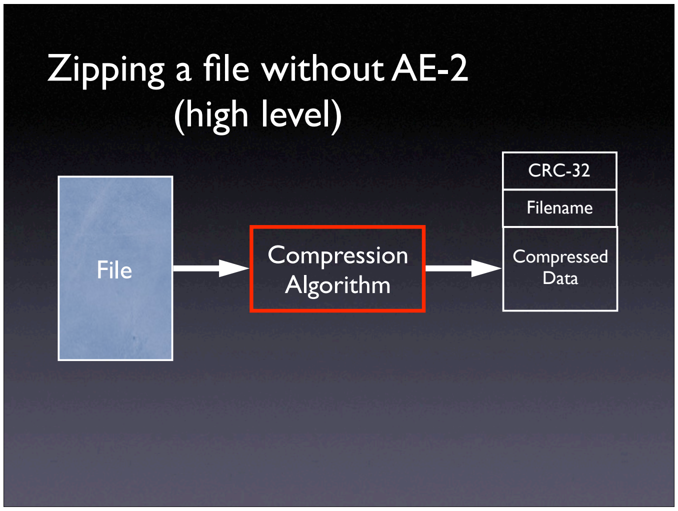

Zipping a file without AE-2
(high level)
File

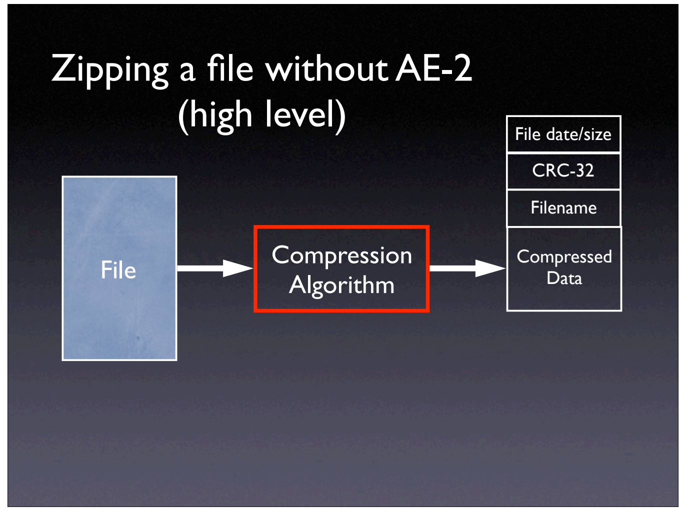

Zipping a file without AE-2
(high level)
File
Compression Algorithm

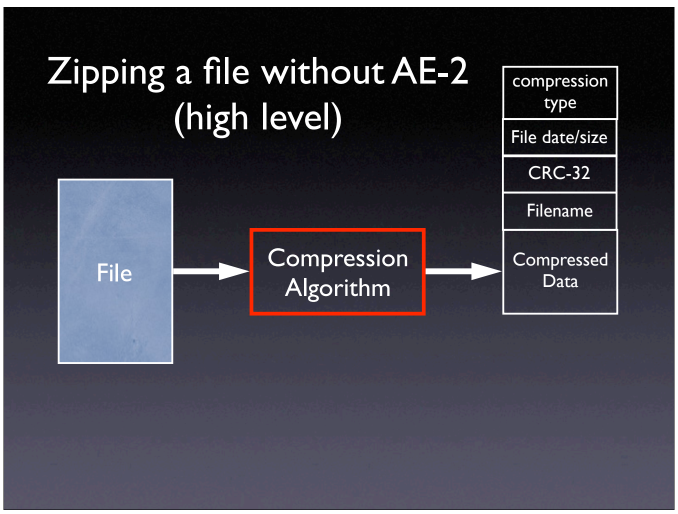

Zipping a file without AE-2
(high level)
File
Compression Algorithm

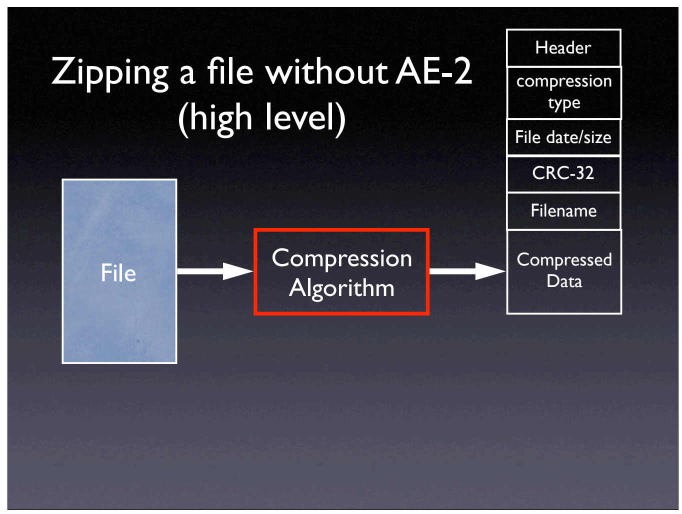

Zipping a file without AE-2
(high level)
File
Compression Algorithm
Compressed Data

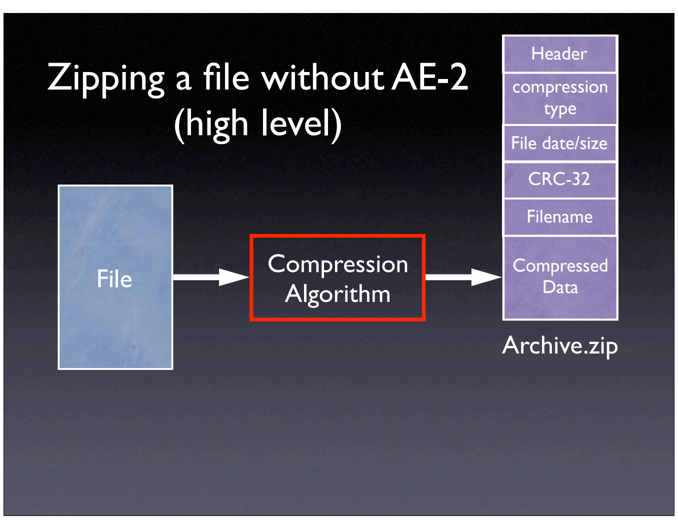

Zipping a file without AE-2
(high level)
File
Compression Algorithm
Filename
Compressed Data

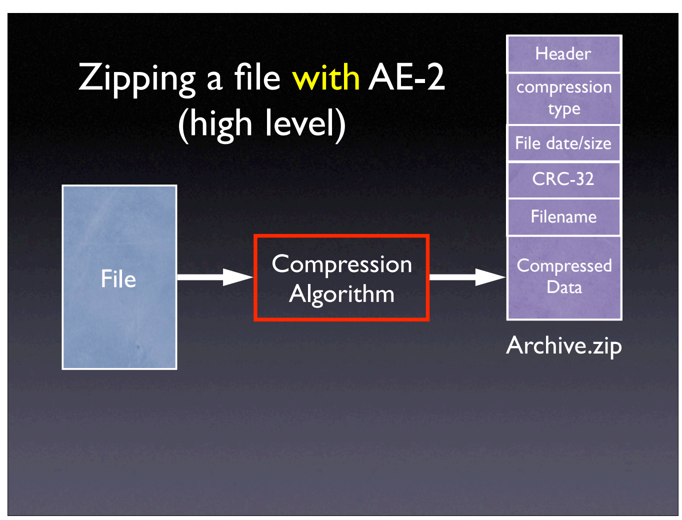

Zipping a file without AE-2
(high level)
File
Compression Algorithm
CRC-32
Filename
Compressed Data

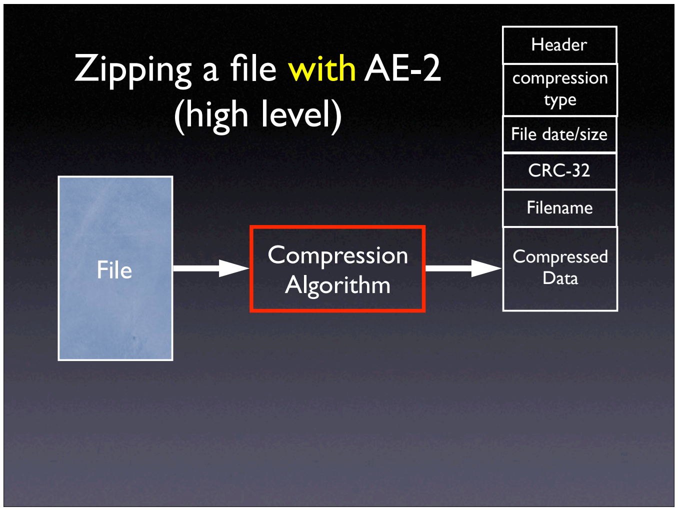

Zipping a file without AE-2
(high level)
File
Compression Algorithm
File date/size
CRC-32
Filename
Compressed Data

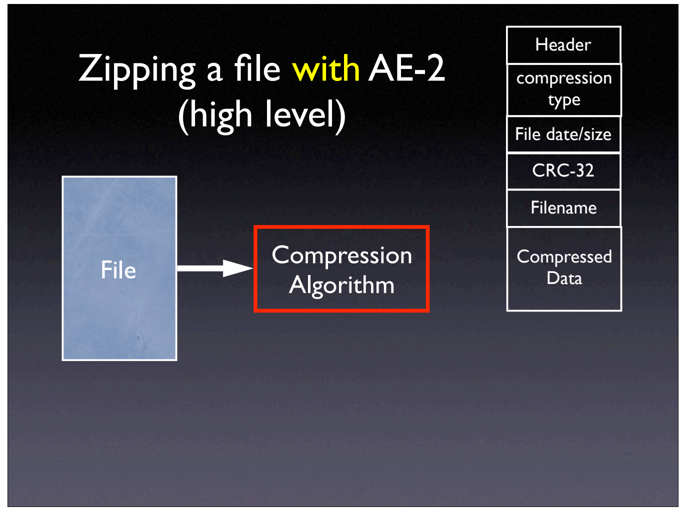

Zipping a file without AE-2
(high level)
File
Compression Algorithm
compression type
File date/size
CRC-32
Filename
Compressed Data

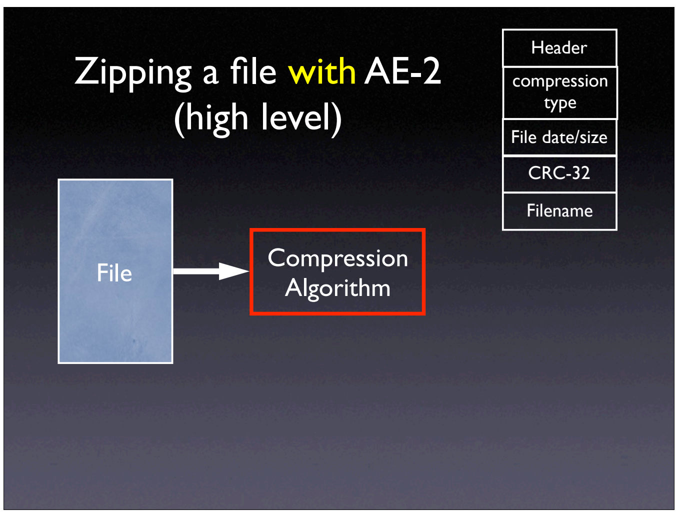

Zipping a file without AE-2
(high level)
File
Compression Algorithm
Header
compression type
File date/size
CRC-32
Filename
Compressed Data

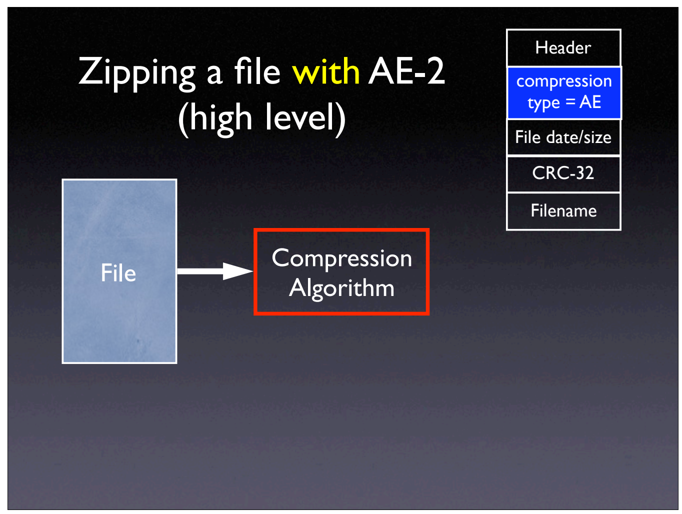

Zipping a file without AE-2
(high level)
File
Compression Algorithm
Header
compression type
File date/size
CRC-32
Filename
Compressed Data
Archive.zip

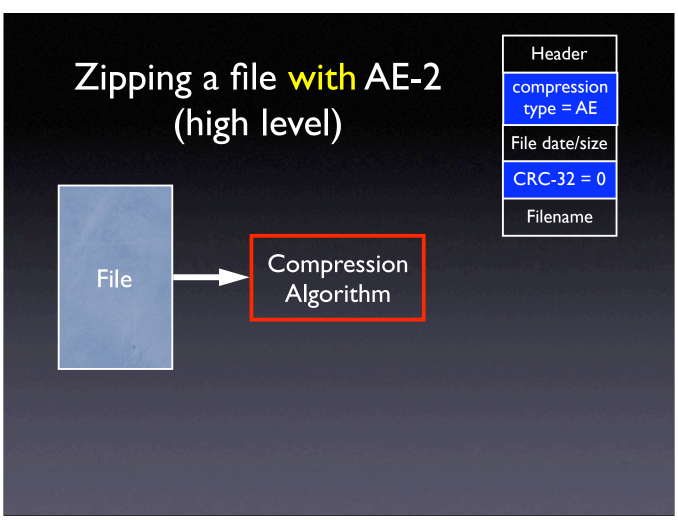

Zipping a file with AE-2
(high level)
File
Compression Algorithm
Header
compression type
File date/size
CRC-32
Filename
Compressed Data
Archive.zip

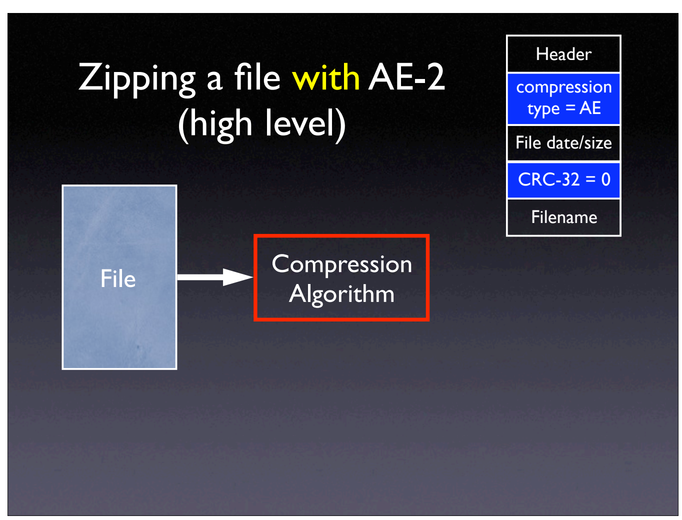

Zipping a file with AE-2
(high level)
Header
compression type
File date/size
CRC-32
Filename
Compressed Data
File
Compression Algorithm

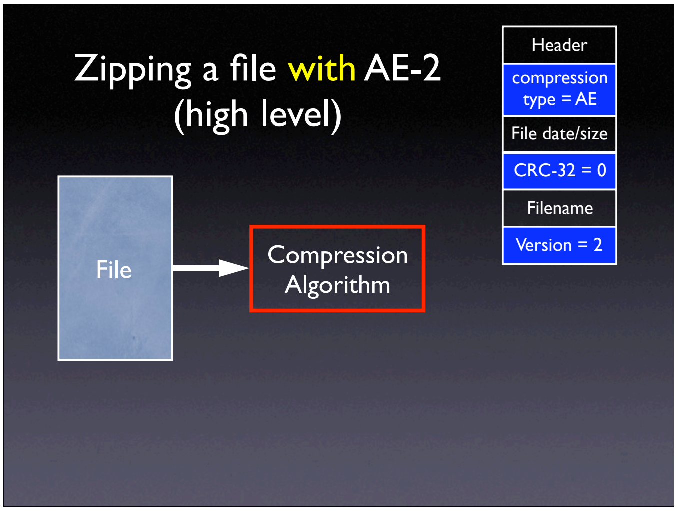

Zipping a file with AE-2
(high level)
Header
compression type
File date/size
CRC-32
Filename
Compressed Data
File
Compression Algorithm

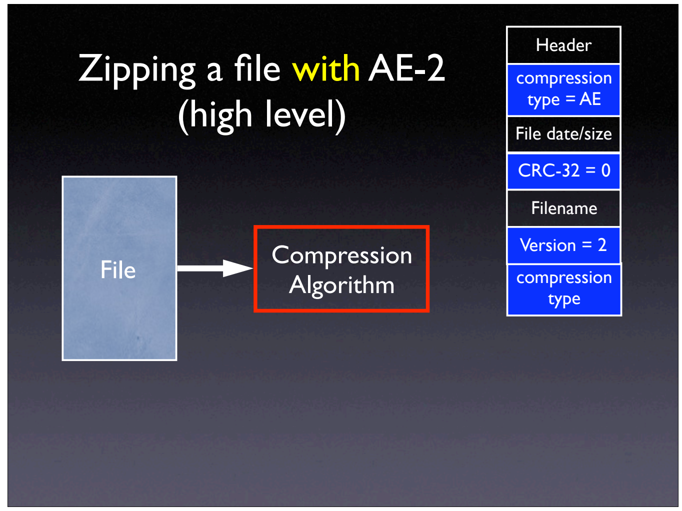

Zipping a file with AE-2
(high level)
Header
compression type
File date/size
CRC-32
Filename
File
Compression Algorithm

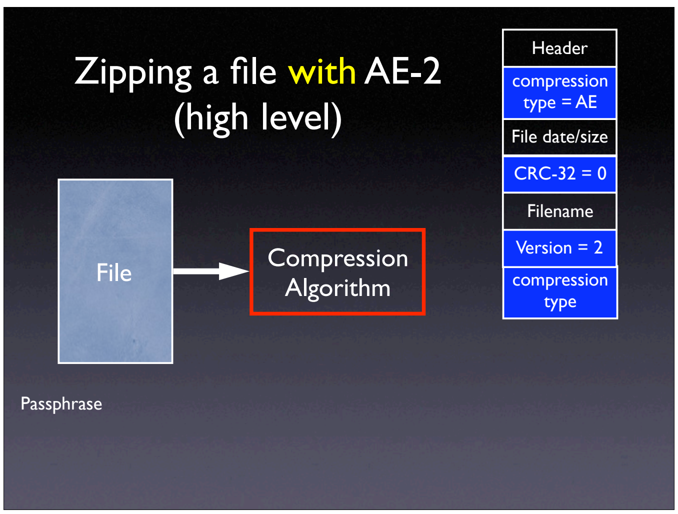

Zipping a file with AE-2
(high level)
Header
compression type = AE
File date/size
CRC-32
Filename
File
Compression Algorithm

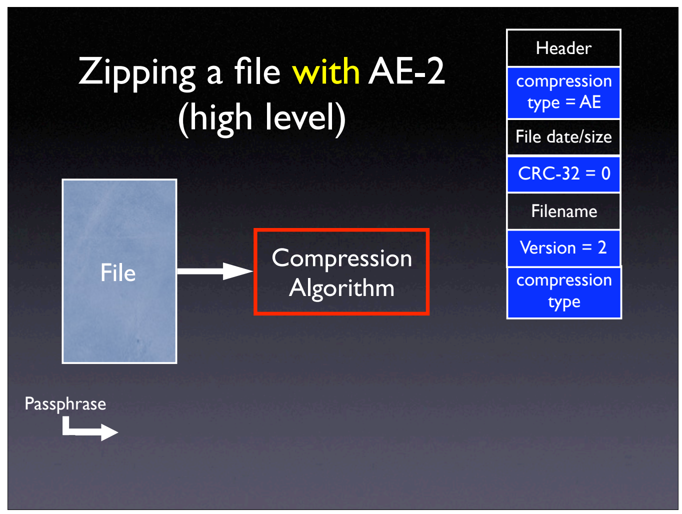

Zipping a file with AE-2
(high level)
Header
compression type = AE
File date/size
CRC-32 = 0
Filename
File
Compression Algorithm

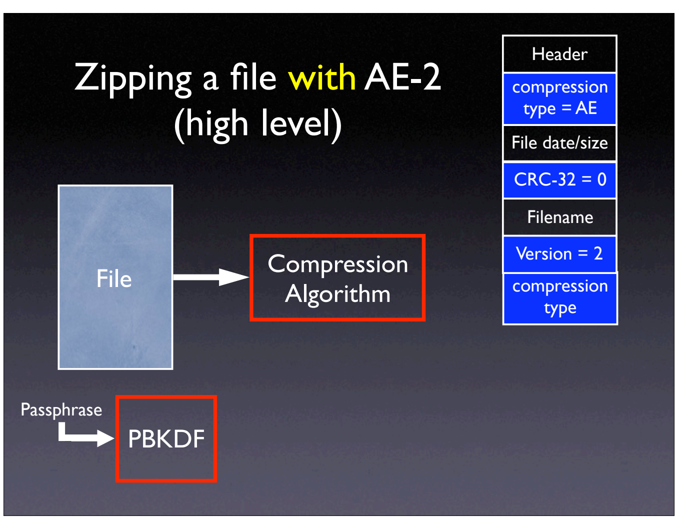

Zipping a file with AE-2
(high level)
Header
compression type = AE
File date/size
CRC-32 = 0
Filename
File
Compression Algorithm

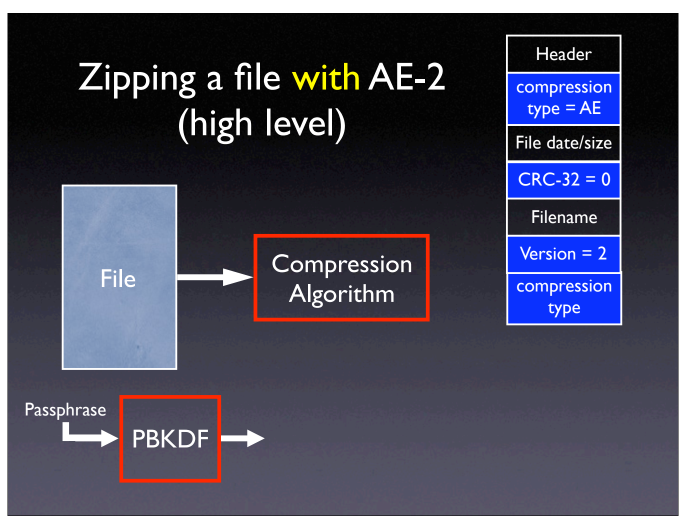

Zipping a file with AE-2
(high level)
Header
compression type = AE
File date/size
CRC-32 = 0
Filename
Version = 2
File
Compression Algorithm

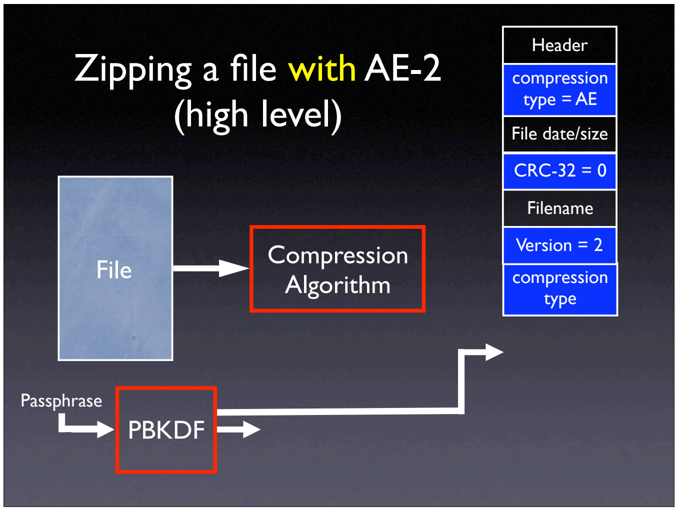

Zipping a file with AE-2
(high level)
Header
compression type = AE
File date/size
CRC-32 = 0
Filename
Version = 2
compression type
File
Compression Algorithm

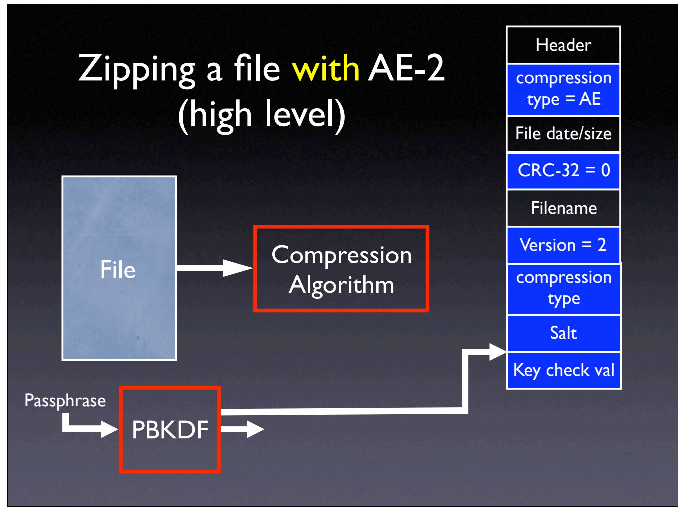

Zipping a file with AE-2
(high level)
Header
compression type = AE
File date/size
CRC-32 = 0
Filename
Version = 2
compression type
File
Compression Algorithm
Passphrase

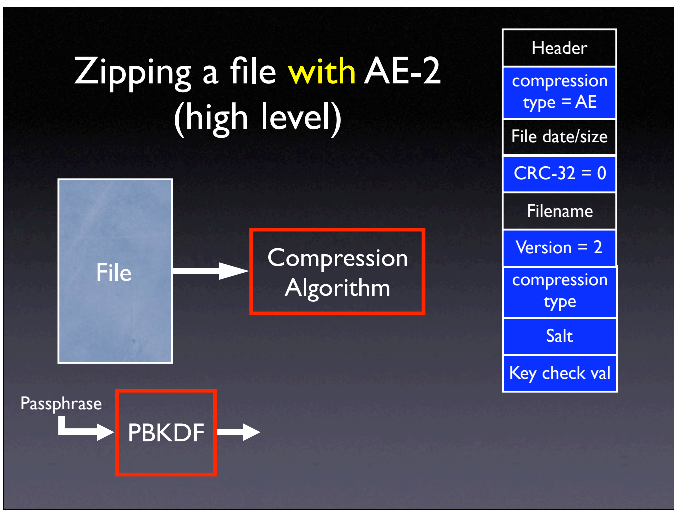

Zipping a file with AE-2
(high level)
Header
compression type = AE
File date/size
CRC-32 = 0
Filename
Version = 2
compression type
File
Compression Algorithm
Passphrase

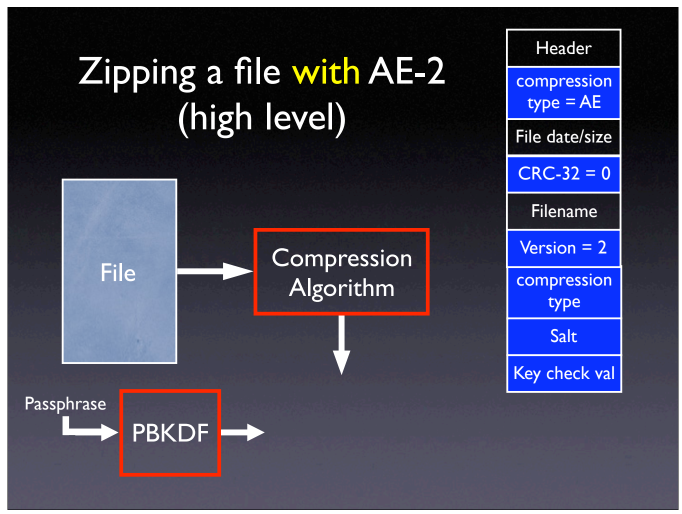

Zipping a file with AE-2
(high level)
Header
compression type = AE
File date/size
CRC-32 = 0
Filename
Version = 2
compression type
File
Compression Algorithm
Passphrase
PBKDF

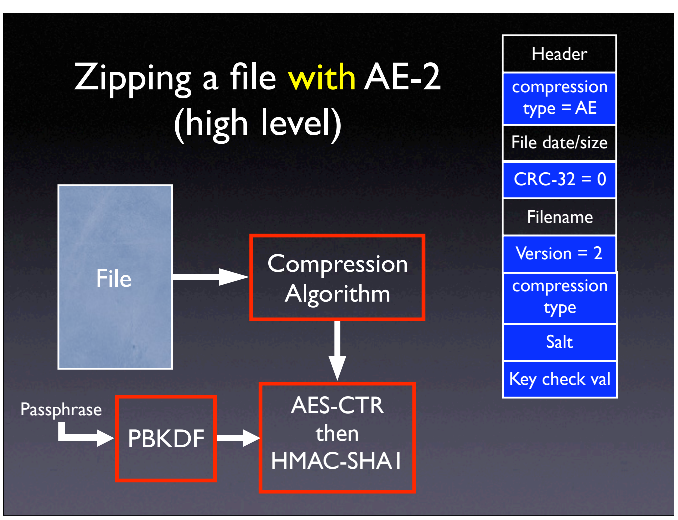

Zipping a file with AE-2
(high level)
Header
compression type = AE
File date/size
CRC-32 = 0
Filename
Version = 2
compression type
File
Compression Algorithm
Passphrase
PBKDF

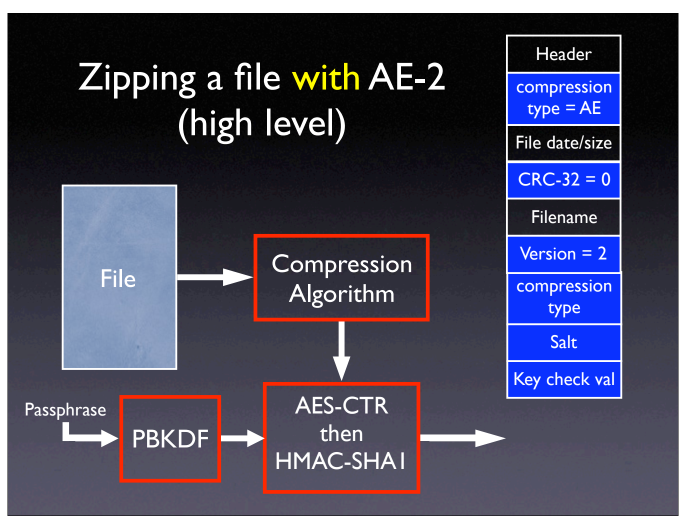

Zipping a file with AE-2
(high level)
Header
compression type = AE
File date/size
CRC-32 = 0
Filename
Version = 2
compression type
File
Compression Algorithm
Passphrase
PBKDF

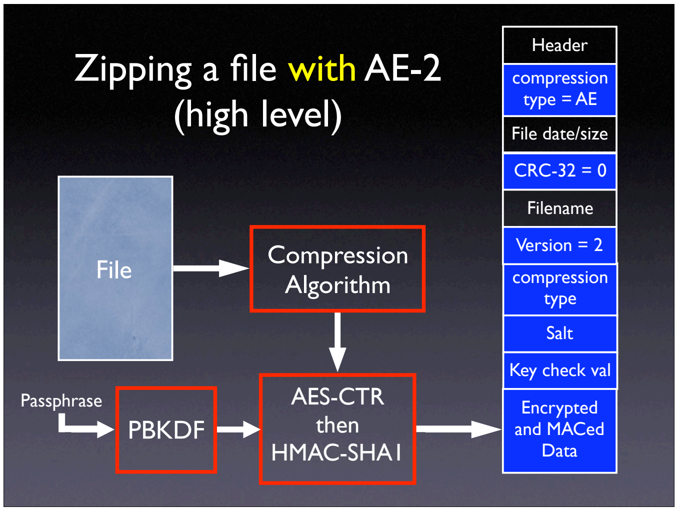

Zipping a file with AE-2
(high level)
Header
compression type = AE
File date/size
CRC-32 = 0
Filename
Version = 2
compression type
Salt
Key check val
File
Compression Algorithm
Passphrase
PBKDF

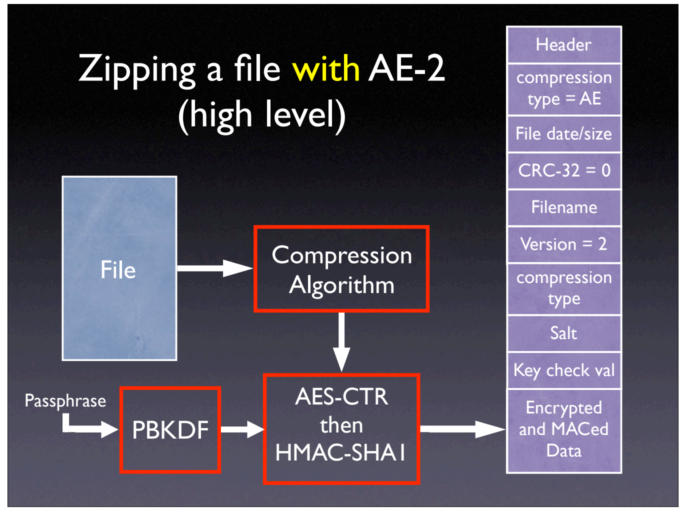

Zipping a file with AE-2
(high level)
Header
compression type = AE
File date/size
CRC-32 = 0
Filename
Version = 2
compression type
Salt
Key check val
File
Compression Algorithm
Passphrase
PBKDF


Zipping a file with AE-2
(high level)
Header
compression type = AE
File date/size
CRC-32 = 0
Filename
Version = 2
compression type
Salt
Key check val
File
Compression Algorithm
Passphrase
PBKDF


Zipping a file with AE-2
(high level)
Header
compression type = AE
File date/size
CRC-32 = 0
Filename
Version = 2
compression type
Salt
Key check val
File
Compression Algorithm
Passphrase
PBKDF
AES-CTR then HMAC-SHA1


Zipping a file with AE-2
(high level)
Header
compression type = AE
File date/size
CRC-32 = 0
Filename
Version = 2
compression type
Salt
Key check val
File
Compression Algorithm
Passphrase
PBKDF
AES-CTR then HMAC-SHA1


Zipping a file with AE-2
(high level)
File
Compression Algorithm
Passphrase
PBKDF
AES-CTR then HMAC-SHA1
Header
compression type = AE
File date/size
CRC-32 = 0
Filename
Version = 2
compression type
Salt
Key check val
Encrypted and MACed Data


Zipping a file with AE-2
(high level)
File
Compression Algorithm
Passphrase
PBKDF
AES-CTR then HMAC-SHA1
Header
compression type = AE
File date/size
CRC-32 = 0
Filename
Version = 2
compression type
Salt
Key check val
Encrypted and MACed Data