Solutions to Selected Problems
Guide to Internet Cryptography

Companion Material

February 17, 2026

Preface This document provides solutions to selected problems from the book *Guide to Internet Cryptography: Security Protocols and Real-World Attack Implications*. The material is intended for educational use in courses and self-study.

Book website: https://link.springer.com/book/10.1007/978-3-031-19439-9

1 Chapter 16: File Encryption: PGP

Problem 16.1 PGP Web of Trust

(a) How many public keys does a public key file typically contain?

(b) What are the key fingerprint and the key ID, and on which key are they computed?

(c) What are typical user identities? Can a user have more than one identity?

Solution

(a) Two. One signature verification key, and one encryption subkey.

(b) The key fingerprint is the SHA-1 hash value of the public signature verification key. The key ID is the last 8 bytes of this hash value.

(c) Internationally standardized digital IDs for end users are rare, mostly international telephone numbers or email addresses. So a typical user ID bound to a signature verification key will be an email address.

Problem 16.2 OpenPGP public key files

Check RFC 4880 for the `Sig type` 0x013 and 0x018. Which parts of **??** do these signature types sign?

Solution

```
0x13: Positive certification of a User ID and Public-Key packet.
    The issuer of this certification has done substantial
    verification of the claim of identity.

0x18: Subkey Binding Signature
    This signature is a statement by the top-level signing key that
    indicates that it owns the subkey.  This signature is calculated
```

```
        directly on the primary key and subkey, and not on any User ID or
        other packets.  A signature that binds a signing subkey MUST have
        an Embedded Signature subpacket in this binding signature that
        contains a 0x19 signature made by the signing subkey on the
        primary key and subkey.
```

0x13 signatures sign the Public Key Packet (tag 6), the User ID Packet (tag 13), and the Hashed Subpackets in the Signature Packet.
0x18 signatures sign the Public Key Packet (tag 6), the Public Key Subpacket (tag 14), and the Hashed Subpackets in the Signature Packet.

Problem 16.3 Web of Trust, TOFU, and Autocrypt

The PGP Web of Trust is based on the transitivity of trust: If A trusts B, and B has signed the public key pk_C of C, then A should believe that pk_C is the public key of C, and that C is trustworthy. In Trust on First Use (TOFU), user A believes that the first step in a process is trustworthy – i.e., that all PGP keyservers can be trusted – and only verifies later actions cryptographically. How does Autocrypt relate to these two concepts? Is the Web of Trust used here, TOFU, or something else?

Solution

Autocrypt is neither of those. A PGP public key contained in an Autocrypt header may overwrite public keys embedded in a Web of Trust, destroying this web. And since each Autocrypt header may contain a different public key for the same email address, the mechanisms also undermines TOFU, since it may always be 'first use'.

Problem 16.4 OpenPGP

(a) How many OpenPGP packets are nested in a ciphertext when hybrid encryption is used?
(b) In hybrid encryption, where are the encryption algorithms specified?

Solution

(a) Packets nested in a ciphertext When hybrid encryption is used in OpenPGP (RFC 4880), the following packets are nested:
1. Public-Key Encrypted Session Key (PKESK) Packet — Tag 1 This packet contains the **session key encrypted with the recipient's public key** (asymmetric step). There is one such packet per recipient. It specifies:

- The recipient's Key ID
- The public-key algorithm used
- The encrypted session key (e.g. RSA- or ElGamal-encrypted)

2. Symmetrically Encrypted (Integrity Protected) Data Packet — Tag 9 or Tag 18 This is the actual ciphertext of the message, encrypted with the **symmetric session key** (symmetric step). It contains, when decrypted:

3. Literal Data Packet — Tag 11 The actual plaintext message body, nested *inside* the encrypted data packet. It carries the filename, date, and the raw message bytes.

Nesting Structure The full nesting structure is as follows:

```
PKESK packet (Tag 1)                    <- asymmetric layer
Symmetrically Encrypted Data (Tag 18) <- symmetric layer
    +-- Literal Data packet (Tag 11)  <- plaintext payload
```

In total, there are **3 packets** (or more if there are multiple recipients, each adding one additional PKESK packet).

(b) Where the encryption algorithms are specified The two algorithms are specified in **different fields** in Table 1.

Important Note A subtle but important point: the symmetric algorithm ID is **not** stored in the Symmetrically Encrypted Data packet (Tag 18) in plaintext. Instead, it is hidden inside the asymmetrically encrypted session key blob in the PKESK packet. Concretely, before asymmetric encryption, the session key material is prepended with a **one-octet symmetric algorithm identifier**:

$$\underbrace{\texttt{sym\_alg\_id}}_{\text{1 octet}} \,\|\, \underbrace{\texttt{session\_key}}_{\text{key material}} \xrightarrow{\text{encrypt with public key}} \underbrace{\texttt{encrypted\_blob}}_{\text{stored in PKESK}} \qquad (1)$$

This blob is only decryptable by the recipient using their **private key**, after which both the symmetric algorithm and the session key become known, allowing decryption of the Tag 18 payload.

Algorithm	**Type**	**Specified in**
Asymmetric algorithm (e.g. RSA, ElGamal, ECDH)	Public-key encryption of session key	PKESK packet (Tag 1), in the *Public-Key Algorithm* octet field
Symmetric algorithm (e.g. AES-256, Camellia)	Bulk encryption of the message	PKESK packet (Tag 1), encoded *within* the encrypted session key MPI

Table 1: Encryption algorithm locations in OpenPGP hybrid encryption.

Problem 16.5 Klima-Rosa small subgroup attack

(a) Why does DSA use two prime numbers, p and q? Would DSA also work with safe primes p?

(b) Would the attack also work if q was changed?

(c) Try to formulate an attacker model for this attack.

Solution

(a) The larger prime number p is used to protect against Index-Calculus algorithms, the smaller to protect against generic attacks like Baby-Step-Giant-Step. Using $q \approx 2^{160}$ gives generic attacks a complexity of about 2^{80}, while reducing the signature size to $2 \cdot 160 = 320$ bits. Using safe prime $p = 2q + 1$, where q is a prime number, would work, too, but then the signatures would become much larger, about the size of p.

(b) When only q is changed, we are in a different category of attacks, the small subgroup attacks from section 12.7.3 against ElGamal KEMs. In the context of DSA, using a smaller value q would facilitate generic attacks, but would probably loose information on x. And any change in the size of q would be immediately be visible in the size of the DSA signature.

(c) The attacker model could be called 'Man-in-the-signature-oracle': The attacker can influence the computation of digital signatures, and can observe the effects from the outside by working with the bad signatures.

Problem 16.6 Klima-Rosa RSA fault attack

(a) Why can the private RSA key be changed without decrypting it first?

(b) How does the extended Euclidean algorithm work?

(c) How many multiplications are necessary to compute a digital signature with the Chinese Remainder Theorem, and how many to compute it with the OpenPGP variant?

Solution

(a) Because of the malleability of the cipher mode used.

(b) Extended Euclidean Algorithm The extended Euclidean algorithm computes, given two integers a and b, not only their greatest common divisor $\gcd(a, b)$, but also the **Bézout coefficients** s and t such that:

$$s \cdot a + t \cdot b = \gcd(a, b) \tag{2}$$

This is especially useful for computing **modular inverses**: if $\gcd(a, n) = 1$, then $s \cdot a \equiv 1 \pmod{n}$, so $s = a^{-1} \bmod n$.

Procedure At each step, the algorithm maintains two equations of the form $s_i \cdot a + t_i \cdot b = r_i$, where r_i is the current remainder. Starting values are:

$$r_0 = a, \quad s_0 = 1, \quad t_0 = 0 \tag{3}$$
$$r_1 = b, \quad s_1 = 0, \quad t_1 = 1 \tag{4}$$

At each step, compute the quotient $q_i = \lfloor r_{i-1}/r_i \rfloor$ and update:

$$r_{i+1} = r_{i-1} - q_i \cdot r_i \tag{5}$$
$$s_{i+1} = s_{i-1} - q_i \cdot s_i \tag{6}$$
$$t_{i+1} = t_{i-1} - q_i \cdot t_i \tag{7}$$

The algorithm terminates when $r_{i+1} = 0$, at which point $r_i = \gcd(a, b)$ and s_i, t_i are the Bézout coefficients.

Example: $\gcd(160, 3)$ **and** $3^{-1} \bmod 160$ See Table 1.
From step 2: $1 \cdot 160 + (-53) \cdot 3 = 1$, so:

$$3^{-1} \equiv -53 \equiv 107 \pmod{160} \quad ✓ \tag{8}$$

(c) Multiplications: CRT vs. OpenPGP Variant

Standard CRT-based RSA Signature To compute $s = H(m)^d \bmod n$ using CRT, the procedure is:

$$s_p = H(m)^{d_p} \bmod p, \qquad d_p = d \bmod (p-1) \tag{9}$$
$$s_q = H(m)^{d_q} \bmod q, \qquad d_q = d \bmod (q-1) \tag{10}$$

Each modular exponentiation with a $k/2$-bit exponent (where n is k bits) using **square-and-multiply** requires approximately:

$$\frac{3}{2} \cdot \frac{k}{2} = \frac{3k}{4} \text{ multiplications} \tag{11}$$

Two such exponentiations are needed (one mod p, one mod q), plus a small number of multiplications for the CRT recombination step:

$$s = s_p + p \cdot (q^{-1}(s_q - s_p) \bmod q) \tag{12}$$

So the **total for standard CRT** is approximately:

$$2 \cdot \frac{3k}{4} + \mathcal{O}(1) = \frac{3k}{2} \text{ multiplications} \tag{13}$$

compared to $3k$ multiplications without CRT — giving a **speedup of roughly** $4\times$.

OpenPGP Variant The OpenPGP variant avoids the full CRT recombination. Instead it uses a **simplified Garner's algorithm**, precomputing $u = p^{-1} \bmod q$ once at key generation time, and then at signing time computing:

$$h = u \cdot (s_q - s_p) \bmod q \tag{14}$$
$$s = s_p + h \cdot p \tag{15}$$

This recombination requires only **2 additional multiplications** on top of the two half-size exponentiations, which is essentially the same asymptotic cost as standard CRT but with a **simpler and slightly more efficient** recombination.

Summary Table (Table 1) Both CRT variants achieve the same $4\times$ **speedup** over naive computation. The OpenPGP variant differs not in the number of multiplications but in the **simpler recombination formula**, which avoids intermediate large-integer arithmetic and is easier to implement correctly.

Step	q	r	s	t
0	—	160	1	0
1	—	3	0	1
2	53	1	1	-53
3	3	0	—	—

Table 2: Extended Euclidean Algorithm applied to $\gcd(160, 3)$.

Method	**Exponentiations**	**Exponent size**	**Total multiplications**
Naive (no CRT)	1	k bits	$\approx \dfrac{3k}{2}$
Standard CRT	2	$k/2$ bits each	$\approx \dfrac{3k}{4} \times 2 = \dfrac{3k}{2} \cdot \dfrac{1}{2}$, i.e. $4\times$ faster
OpenPGP CRT	2	$k/2$ bits each	$\approx \dfrac{3k}{4} \times 2 + 2 \approx \dfrac{3k}{2} \cdot \dfrac{1}{2}$

Table 3: Comparison of multiplication counts for RSA signature computation.