El Gamal Cryptosystem

P=23, generator g=5, Private key d=6, Message M=10. (Private key can be taken as d or x)

(a) compute the public key

$Y = g^d \bmod p$

$= 5^6 \bmod 23$

$\boxed{Y = 8}$

(b) choose K =4 and compute C_1

$C_1 = g^K \bmod p$

$= 5^4 \bmod 23$

$\boxed{C_1 = 4}$

(c) compute C_2

$C_2 = M * Y^K \bmod p$

$= 10 * (8^4) \bmod 23$

$C_2 = 20$

(d) Decrypt M and verify M=10

$M = C_2 * (C_1^{-x}) \bmod p$ (x is private key)

$= 20 * (4^{-6}) \bmod 23$

$\boxed{M = 10}$

'M' has been recovered and it is '10'

(e) Explain why using a different k for the same M produces a different cipher text.

* Because both C_1 and C_2 depends on K.

In C_1 we are using in $g^K \bmod p$ and In C_2 we are using $M * Y^k \bmod P$ that's why changing K will give differ cipher text.

Symmetric key distribution using Asymmetric Encryption

P=53 generator (α) g=2
Alice's private key a=8
Bob's private key b=11

(a) compute Alice's public Key

$A = g^a \bmod p$

$= 2^8 \bmod 53$

A=44

(b) compute Bob's public Key

$B = g^b \bmod p$

$= 2^{11} \bmod 53$

B=34

(c) compute shared session key for Alice

$K = B^a \bmod p$

$= 34^8 \bmod 53$

K=36

(d) compute the shared session key for Bob

$K_B = A^b \bmod p$

$= 44^{11} \bmod 53$

$\boxed{K_B = 36}$

(e) Verify both sides get the same K, and Explain how K is then used to encrypt actual messages symmetrically.

Both Alice & Bob Shared session key K = 36 , It is verified.

* The K is converted to a strong sym-Key using Key Derivation function.

* That Key will be used to Encrypt plain texts to cipher text using Any Symmetric based Cipher Algorithms.

1) Digital Signature Standard.

P = 31, q = 5, g = 2, Private key x = 3,
K = 4 and Hash H(M) = 6.

(a) compete the public key

$\Rightarrow Y = g^x \bmod p$

$= 2^3 \bmod 31$

$= 8 \bmod 31$

$\boxed{Y=8}$

(b) compute 'r'

$r = (g^k \bmod p) \bmod q$

$= (2^4 \bmod 31) \bmod 5$

$= (16 \bmod 31) \bmod 5$

$= 16 \bmod 5$

$\boxed{r=1}$

(c) compute 's'

$S = k^{-1}(H(M) + x \cdot r) \bmod q$ (where x=3)

$= 4^{-1}(6 + 3 \cdot 1) \bmod 5$

$= 4^{-1}(9 \bmod 5)$

$= (4^{-1} \bmod 5)(9 \bmod 5)) \bmod 5$

$= (4 \cdot 4) \bmod 5$

$= 16 \bmod 5$

$\boxed{S=1}$

Side Calculation (for k^{-1})

Subs x=4

$4x=1 \bmod 5$

$4(4) \bmod 5 = 1$

(d) verify the signature by computing u, u1, u2, v

$w = s^{-1} \bmod q$

$u1 = H(M) * w \bmod q$

$u2 = r * w \bmod q$

$v = (g^{u1} * y^{u2} \bmod p) \bmod q$

$w = 1^{-1} \bmod 5 \Rightarrow 1x \equiv 1 \bmod 5 \Rightarrow x = 1$

$x = 6$

$w = 1$

$u1 = 6 * w \bmod 5$

$= 6 * 1 \bmod 5 \Rightarrow 1$ $u1 = 1$

$u2 = 1 * w \bmod 5$

$= 1 * 1 \bmod 5 \Rightarrow 1$ $u2 = 1$

$v = (g^1 * y^1 \bmod 31) \bmod 5$

$= (2 * 8 \bmod 31) \bmod 5$

$= (16 \bmod 31) \bmod 5$

$= 16 \bmod 5$

$v = 1$

(e) Confirm v = r to complete verification, and explain why k must be kept secret and never reused

v = 1 and r = 1

v = r this signature is valid.

*If an attackes Can able to get the K value then it is easy for them to Calculate Private key x.

*Once the private key is exposed then the attacker can Create Valid Signatures.

* That is why 'k' value must kept secret and never reused.

Kerberos Authentication System

Kerberos is a network authentication protocol that uses secret-key cryptography to provide strong authentication for client-server applications.

Key Components

- **Client (C)**: The user or service requesting authentication
- **Authentication Server (AS)**: Verifies the client's identity using a shared secret key
- **Ticket Granting Server (TGS)**: Issues service tickets after AS authentication
- **Service Server (SS)**: The actual service the client wants to access
- **Key Distribution Center (KDC)**: Combines AS and TGS

Kerberos Authentication Steps

Step 1: AS Request (Client → AS)

Client sends: {Client ID, TGS ID, Timestamp} to the Authentication Server.

Step 2: AS Response (AS → Client)

AS verifies client identity from its database and sends:

- **TGT (Ticket Granting Ticket)**: {Client ID, TGS ID, Timestamp, Lifetime, Session Key $K_{c,tgs}$} encrypted with TGS's secret key
- **Message encrypted with Client's secret key**: {Session Key $K_{c,tgs}$, TGS ID, Timestamp, Lifetime}

Step 3: TGS Request (Client → TGS)

Client decrypts the message using its secret key and sends:

- **TGT** (forwarded as received)
- **Authenticator**: {Client ID, Timestamp} encrypted with Session Key $K_{c,tgs}$
- **Service Server ID**

Step 4: TGS Response (TGS → Client)

TGS issues a Service Ticket and sends:

- Service Ticket: {Client ID, SS ID, Timestamp, Lifetime, Session Key $K_{c,ss}$} encrypted with SS's secret key
- Message encrypted with $K_{c,tgs}$: {Session Key $K_{c,ss}$, SS ID, Timestamp, Lifetime}

Step 5: Service Request (Client → SS)

Client sends to the Service Server:

- Service Ticket (forwarded as received)
- Authenticator: {Client ID, Timestamp} encrypted with $K_{c,ss}$

Step 6: Mutual Authentication (SS → Client)

SS decrypts the ticket, verifies the authenticator, and returns:

{Timestamp + 1} encrypted with $K_{c,ss}$ — proving mutual authentication is complete.

Advantages of Kerberos

- Single Sign-On (SSO): Login once, access multiple services
- No password sent over network — only encrypted tickets
- Tickets have limited lifetime — reduces replay attack risk
- Mutual authentication — both client and server are verified

Limitations of Kerberos

- KDC is a single point of failure — if KDC is down, no authentication
- Requires synchronized clocks — timestamp-based validation

Vulnerable to password guessing if weak passwords are used

X.509 Certificate Structure and Public Key Distribution

X.509 is a standard format for digital certificates used to verify identities and distribute public keys securely over the internet. It acts like a digital passport that proves authenticity in online communications.

(a) Five Key Fields of an X.509 Certificate

1. Validity Period
- Contains two timestamps: 'Not Before' and 'Not After'
- Defines the time window when the certificate is valid
- Example: Valid from January 1, 2024 to January 1, 2025

2. Subject Public Key
- Contains the public key and its algorithm (RSA, ECDSA, etc.)
- This is the key used for encryption and digital signature verification

3. Signature Algorithm
- Specifies the cryptographic algorithm used to sign the certificate
- Common examples: SHA-256 with RSA, SHA-384 with ECDSA
- Ensures certificate integrity and authenticity

4. Certificate Authority (Issuer)
- Identifies the trusted organization that issued and signed the certificate
- Acts as a trusted third party vouching for the certificate holder
- Examples: DigiCert, Let's Encrypt, GlobalSign

5. Extensions
- Optional fields providing additional certificate information
- Extended Key Usage: Specifies purposes like web server authentication or email protection.

(b)

Chain of Trust

The chain of trust is a hierarchical structure of certificates that establishes credibility.

Root CA (Top Level)

- Self-signed certificate (signs its own certificate)
- Pre-installed in operating systems and browsers
- Highly secured and rarely used directly for signing

Intermediate CA (Middle Level)

- Signed by the Root CA
- Handles day-to-day certificate issuance
- Provides isolation — if compromised, only this level needs revocation

End-Entity Certificate (Bottom Level)

- Your actual certificate (for website, email, etc.)
- Signed by an Intermediate CA
- Used directly for secure communications

Verification Process

- Browser receives the end-entity certificate
- Checks if it's signed by a trusted Intermediate CA
- Verifies the Intermediate CA is signed by a Root CA

(c)

Certificate Revocation List (CRL)

A CRL is a digitally signed list of certificates that have been revoked before their expiration date and should no longer be trusted.

- CA Maintains List: Certificate Authority keeps a database of revoked certificates
- Regular Publication: CRL is published at a URL specified in the certificate
- Client Downloads: Applications periodically fetch the latest CRL
- Status Check: Before trusting a certificate, client checks if its serial number appears in CRL
- Rejection: If found on the list, certificate is rejected even if still within validity period

(d)

Real-World Example: TLS/SSL Handshake
Scenario: Visiting https://scholarpeak.in

- Step 1 — Browser sends 'Client Hello' requesting secure connection
- Step 2 — Server sends 'Server Hello' with its X.509 certificate containing its public key
- Step 3 — Browser checks validity period, verifies domain name, validates digital signature, and traces chain of trust to a Root CA
- Step 4 — Browser generates a session key, encrypts it using server's public key, and sends it. Only server (with matching private key) can decrypt it
- Step 5 — Both parties share a secret session key; all subsequent data is encrypted with this key

(e) Two Scenarios for Certificate Revocation

Scenario 1: Private Key Compromise
Example: A company discovers its web server was breached and the private key was stolen.
- Attackers can now impersonate the legitimate server
- Certificate must be revoked instantly
- Contact CA → Generate new key pair → Issue new certificate

★ Real Case: The Heartbleed vulnerability (2014) forced mass certificate revocation when private keys were exposed.

Scenario 2: Change in Certificate Details
Example: A company is acquired and changes its legal name from 'TechStart Inc.' to 'MegaCorp Technologies.'
- Certificate information no longer matches the organization
- Revoke old certificate with outdated information
- Request new certificate with correct company name
- Other cases: domain ownership transfer, employee departure, merger or restructuring