Programming Mobile Devices

Security Aspects

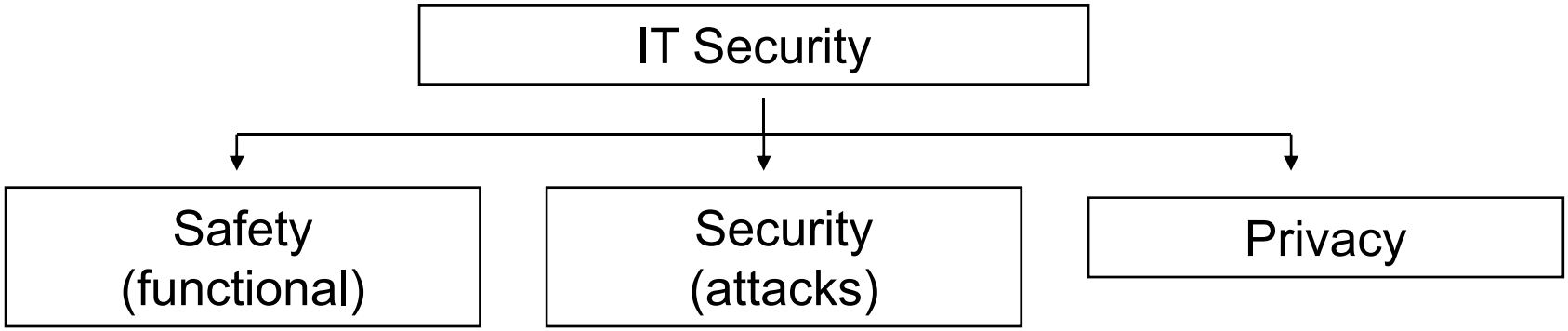
University of Innsbruck

WS 2009/2010

thomas.strang@sti2.at

STI · INNSBRUCK

3 Domains of IT Security



Safety

Protection against un-intentional physical, social, spiritual, financial, political, emotional, occupational, psychological, educational or other types of un-desired consequences of system failures

Typical design methods:

- Fail-Safe
  - in case of failure, fallback into a safe (usually static) state
- Fault-Tolerance
  - in case of failure, continue operation on reduced QoS level (graceful degradation)

Caution: Safety not equal to Reliability / Quality of Service

Security

Protection against intentional attacks, in particular concerning

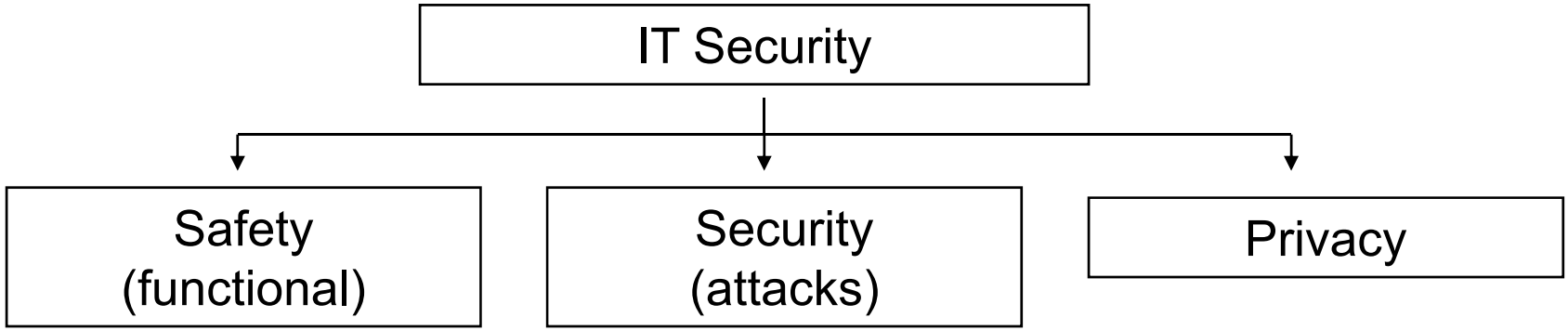
- Confidentiality
- Integrity
- Availability
- Independency
- Non-Observability
- Accountability / Non-Repudiation
- Liability
- Anonymity / Pseudonymity
- Authenticity (as part of AAA: Authentication, Authorization, Accounting)

Privacy

Protection against misuse of information, in particular concerning

- personal data and misuse by third parties
- avoid to harm personal rights

3 Domains of IT Security



→ **enough for a full lecture!**

→ we focus here on some selected cases for PMD…

Authentication

"Alice makes sure that she's talking to Bob, not someone else"

- Three classical options:
  - by knowledge: **"password"**
  - by ownership:

  - by property:

static

semi-static

(Authorization)

- Authorization – giving access to peer
  - “Alice allows Bob to do certain things”
  - Authenticated (known) peer
  - Allowing or blocking an action
  - In a bank – access to your account, not others
- Authorization requires authentication

Typical Improvements of Authentication by Knowledge (Passwords)

- Hashing – host ("Bob") knows only h = H(m)
  - eavesdropper attack: okay
  - playback attack: negative
- Timestamps
  - using trusted third party (e.g. Surety), e.g. based on *Discrete Logarithm Problem* to avoid eveasdropping and playback

    $y = t^h \bmod p$ with t=timestamp, h=H(passwort)
- systematically: Challenge-and-Response protocols

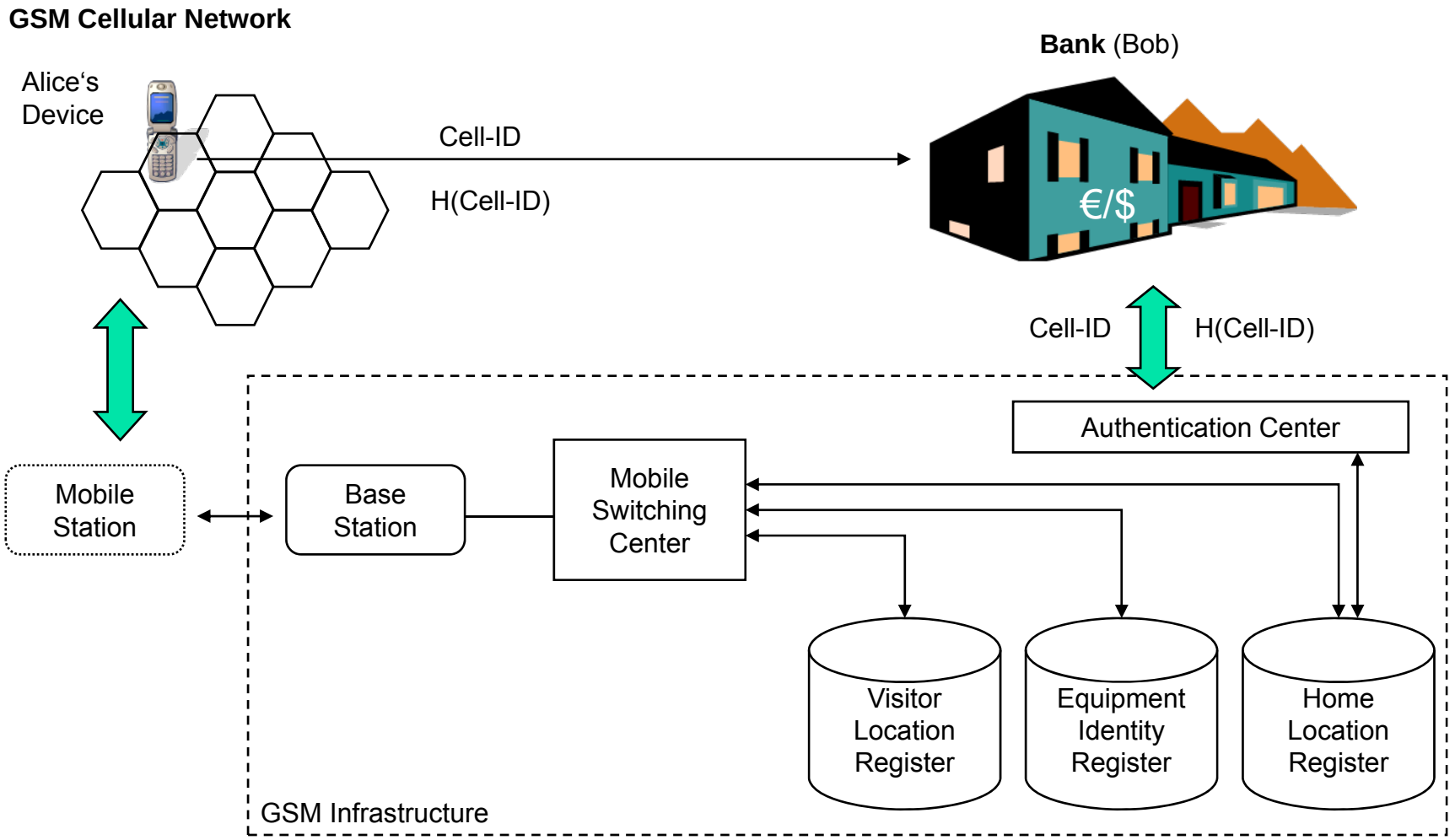
New opportunities by mobile devices

- Usually Alice's knowledge is randomly selected
- Alice should prove knowledge of a dynamic secret each time!

- Penetration of mobile devices makes one information a predominant candidate: The geographic position of Alice.
  - user-dependent dynamics
  - known to user or may be determined easily (GPS, INS, GSM etc.)

- Alice and Bob must know location

Bank Scenario

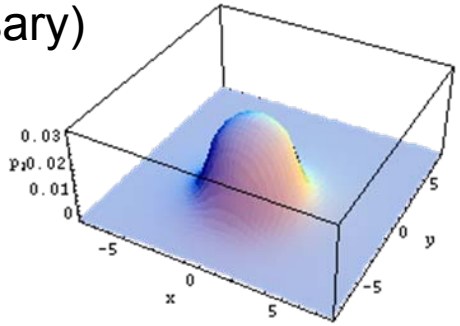


Location-Based Authentication (LBA)

Making Authentication dependent to successful location check!

Sometimes simplifications possible:

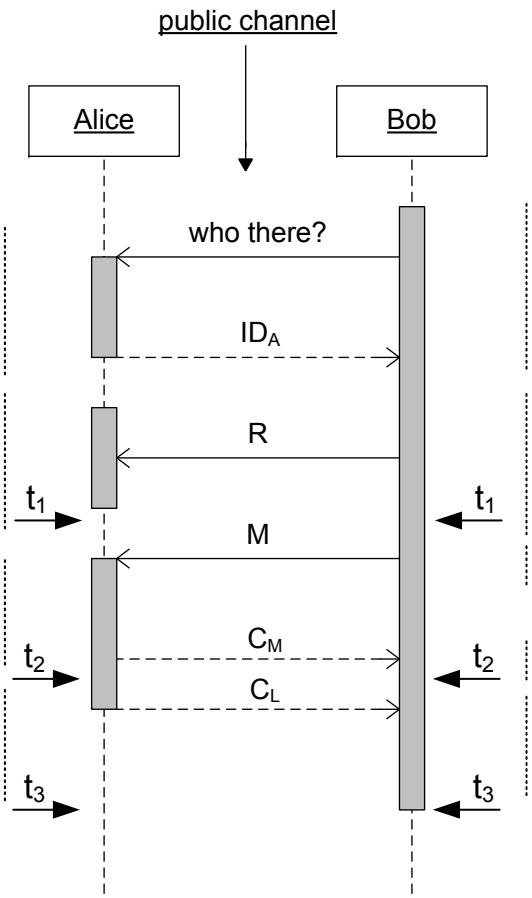
- at least be able to determine distance
- plausibility check (no location DB access for Bob neccessary)
  - movement (SoLo)
  - multiple places in small time window (areas of world)



11:02 - "Alice is in Seattle"

11:04 - "Alice is in Sydney"

Challenge-by-Location and Response Authentication (CLARA) protocol



0: (Init) α, p public
Bob knows (as any other authorized entity) the identity ID_A of Alice and the hashvalue of her password $y := \alpha^{P_A} \bmod p$, but not the password P_A

step 1: Alice knows her ID_A and her P_A

Alice identifies herself to Bob as ID_A

step 2: Bob chooses random Z with $0 <= Z < p$ and calculates $R := \alpha^Z \bmod p$ as well as $K := y^Z \bmod p = (\alpha^{P_A})^Z \bmod p = \alpha^{ZP_A} \bmod p$

step 3: Alice calculates K from R with

$K := R^{P_A} = (\alpha^Z)^{P_A} \bmod p = \alpha^{ZP_A} \bmod p$

step 4: Bob chooses random M with $0 <= M < p$

step 5: Alice calculates C_M from M with K

$C_M := \text{encrypt}(M, K)$

step 6: Bob checks if $M == \text{decrypt}(C_M, K)$

step 7: Alice calculates C_L from her position L with K

$C_L := \text{encrypt}(L, K)$

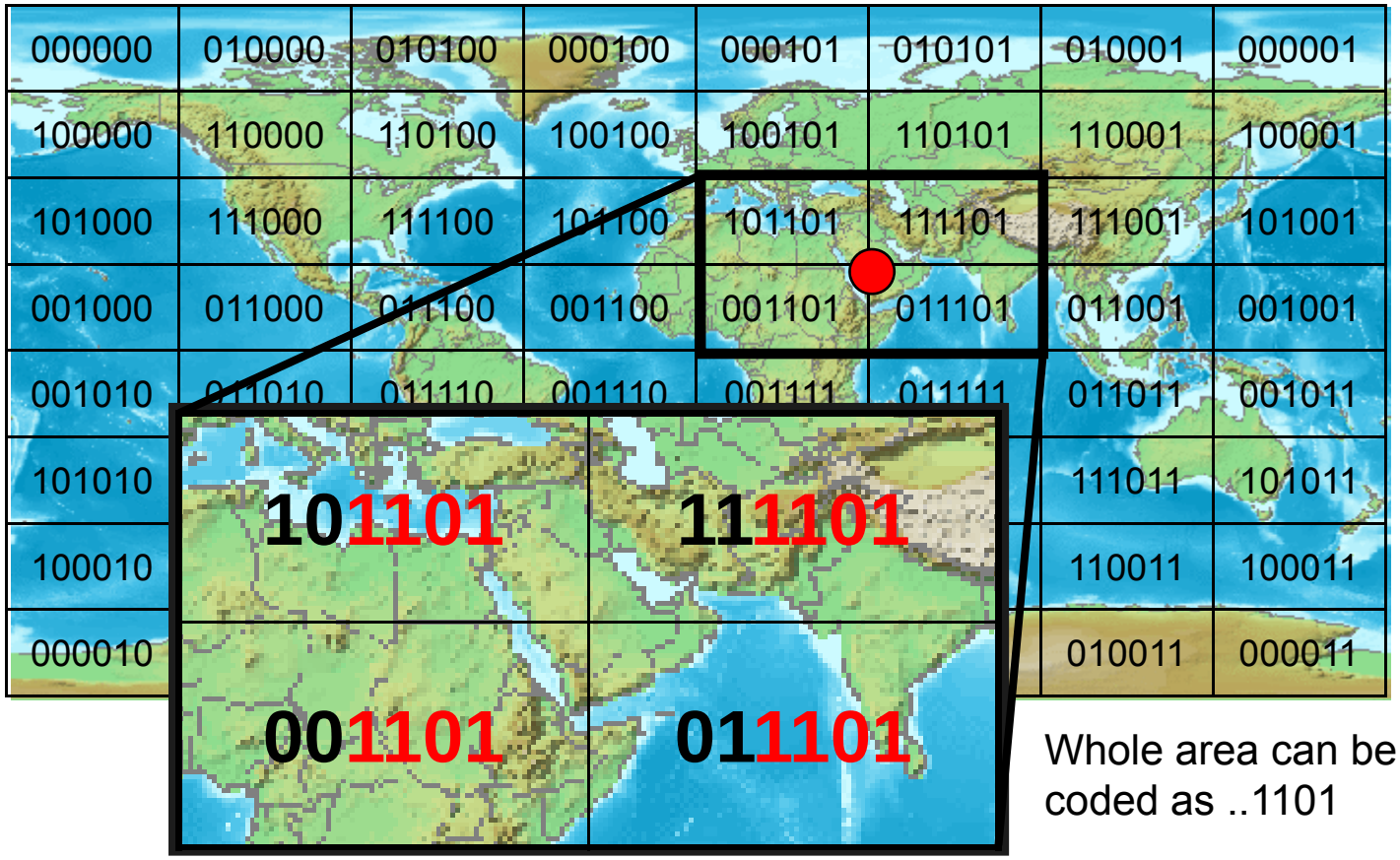
step 8: Bob requests from Location Registry the current position L of ID_A and checks if $L == \text{decrypt}(C_L, K)$

[Strang, 2006 - http://elib.dlr.de/22383/]

Privacy Issues

- Privacy
  - What if Alice does not want to (fully) disclose her location to Bob?
    - Location Hash
    - Hierarchical location encoding (Innsbruck $\subset$ Tirol)
    - Location Blurring (adding artificial errors)
  - Who should be allowed to calculate distance?
    - access schemes

One Solution



[Strang et al..: „Using Gray codes as Location Identifier“, KuVS FG Ort, Bonn, September 2009]

Security in Java SE

- Java Cryptography Extension (JCE, since JDK1.4)
  - http://java.sun.com/javase/technologies/security/
  - `javax.crypto`
- e.g.: Encryption/Decryption

Java SE Security API – Encryption Example

```
KeyGenerator keygen = KeyGenerator.getInstance("DES");
SecretKey desKey = keygen.generateKey();

Cipher desCipher =
        Cipher.getInstance("DES/ECB/PKCS5Padding");

// Initialize the cipher for encryption
desCipher.init(Cipher.ENCRYPT_MODE, desKey);

byte[] cleartext = "This is just an example".getBytes();

// Encrypt the cleartext
byte[] ciphertext = desCipher.doFinal(cleartext);
```

Java SE Security API – Decryption Example

```
// KeyGenerator keygen = KeyGenerator.getInstance("DES");
// SecretKey desKey = keygen.generateKey();

Cipher desCipher =
        Cipher.getInstance("DES/ECB/PKCS5Padding");

// Initialize the cipher for decryption, symmetrical key
desCipher.init(Cipher.DECRYPT_MODE, desKey);

// byte[] ciphertext = /* the ciphertext from previous code */;

// Decrypt the ciphertext
byte[] cleartext1 = desCipher.doFinal(ciphertext);
```

Basic Security in MIDP

- MIDP 2.0
  - Mandates HTTPS
  - Optionally supports Java Security API
  - Does **not** support Java Cryptography Extension, but subset:
    - SATSA-CRYPTO (JSR-117)
  - Alternative: Bouncy Castle API

SATSA

- SATSA-CRYPTO (JSR-117)
  - *message digests*: used to create a "fingerprint" of a piece of data (aka *hashvalue*)
    - algorithms: e.g. SHA-1 or RipeMD
  - *digital signatures:* useful for verifying data integrity, generated using a private key (much harder to forge compared to message digests)
  - *ciphers*: decription/encryption algorithms
    - typically symmetric only

Alternative: Bouncy Castle

- Bouncy Castle
  - **Lightweight** Cryptography API
  - Implementation that works in MIDP/J2ME
  - Supports (asymetric) public-key algorithms
  - particularly useful: `java.math.BigInteger`
  - http://bouncycastle.org/

Exercise 15 – generate session key

- Implement the Diffie-Hellman key exchange protocol using BouncyCastle on a MIDP phone client (Alice) and a J2SE server (Bob)
  - example of Diffie-Hellman usage in MIDP at https://www.datenzone.de/space/BouncyCastle
- Use the Location API to determine the phone's position and use this information as Alice‘s password P_A
- Print the generated session key on both sides (MIDP and J2SE)