GSM

- Global System for Mobile Communications, 1992
- Security in mobile phones
- System used all over the world

GSM: Threat Model

- What
  - Cloning
  - Eavesdropping
  - Tracking
- Who
  - Criminals
  - Secret Services
- Why
  - Break Confidentiality
  - Free phone calls
  - Reveal whereabouts
- How
  - Break Crypto
  - Exploit bad design

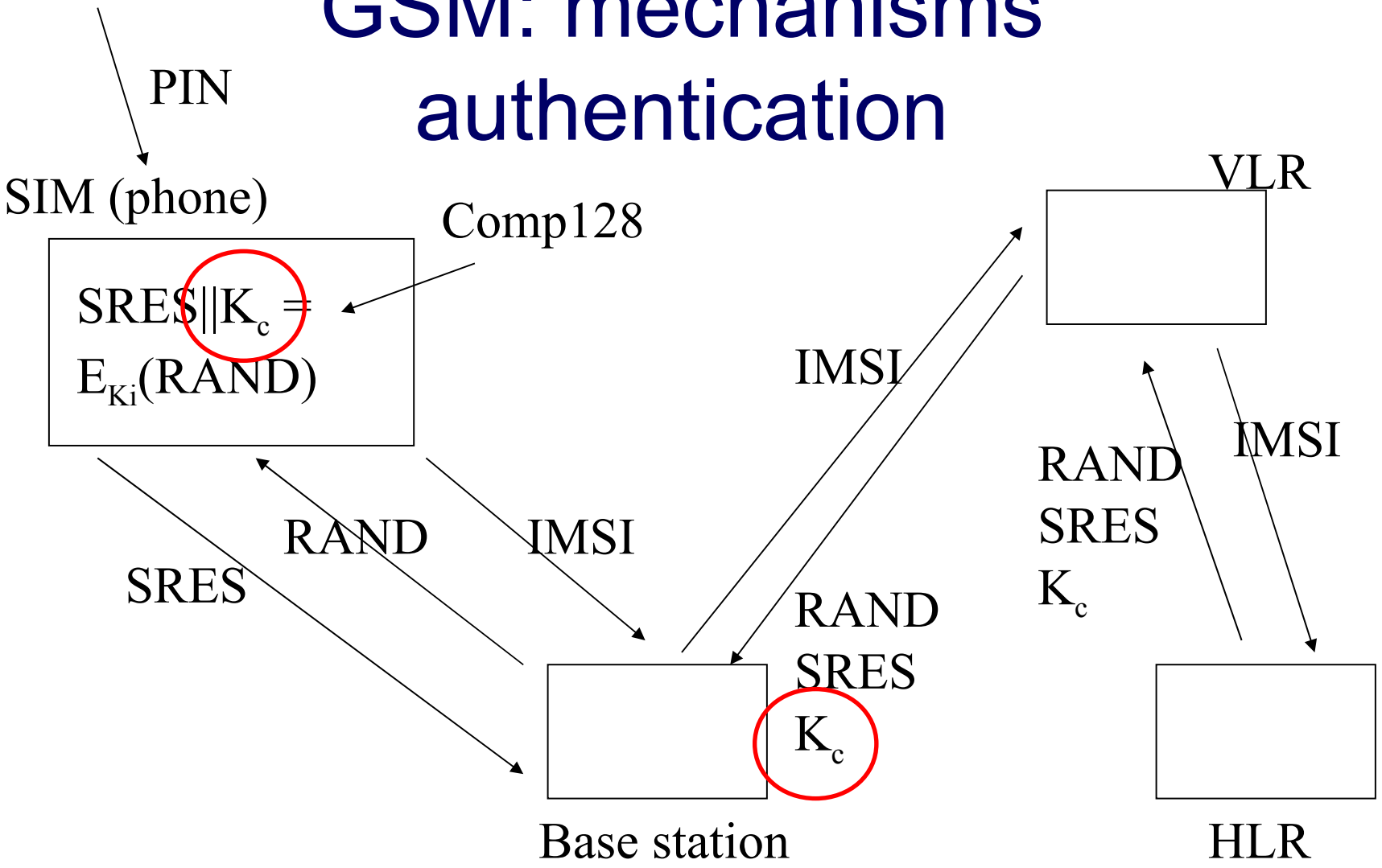
GSM: Security Policy

- Security Objectives
  - Authentication
  - No tracking
  - Confidential Calls
- Strategy
  - Crypto
  - SIM PIN codes

GSM-system

- SIM
  - PIN
  - IMSI
  - K_i
- Base station
- HLR
- VLR

GSM: mechanisms authentication



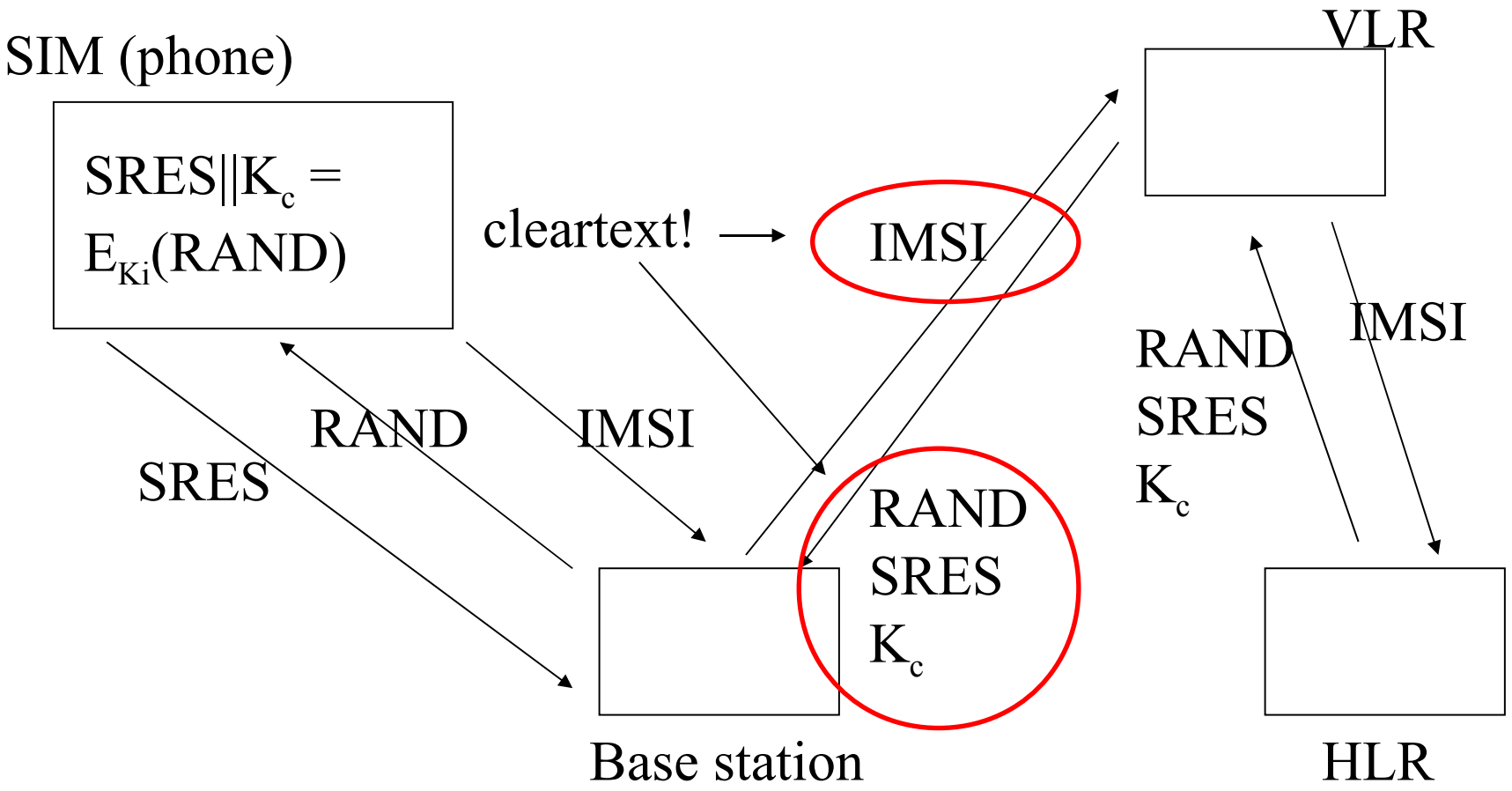
GSM: mechanisms
No tracking

- When SIM registers on network
  - TMSI – temporary/anonymous IMSI
- But IMSI must still be sent initially

GSM: mechanisms Confidentiality

- All conversation encrypted
  - Key: Kc
  - Algoritme: among others, A5 (was secret, like Comp128)

GSM: attack1 on authentication



GSM: attack2 on authentication

- Access to SIM
- 150.000 well chosen challenges
  - Exploit weaknesses in Comp128
  - Find K_i

GSM: attack/tracking

- When SIM registers on network
  - TMSI – temporary/anonymous IMSI
- But IMSI sent initially

- IMSI-catcher
  - Strong signal
  - Pretend not to understand ”forstå” TMSI
  - SIM sends IMSI

GSM: attack on Confidentiality

- All conversation encrypted
  - Key: Kc
  - Algorithm: A5 and others(originally secret, like Comp128)
- A5 and the way it is used has weaknesses
  - Attack can be done within minutes

GSM: what can we learn?

- Krypto the weakest link?!
  - Kerchhoffs principle (Comp128 og A5 secret)
- Misunderstanding of architecture
  - Transmission of keys in cleartext ☹
- Was GSM security a succes or a failure?
  - for who?

Buffer overflows

- Very ”popular” securitybreach
- Microsoft estimates *internal* expense of $100.000 pr. patch
- Problem caused by bad code and languages that do not protect against it
  - C, C++
- Change to Java, C#, …,? Does’t always help, many OS’s are written in C

Stack overruns

```
void foo(char* input){
   char buf[3];
   strcpy(buf, input);
}

void bar(void){
   printf(”Gotcha!”);
}

int main(int argc, char* argv[]) {
   foo(argv[1])
   return 0;
}
```

Compiled program

Addr	Code
0001	main:
0002	push argv[0]
0003	goto foo
0004	pop
0005	goto exit
0006	foo:
0007	allocate buf
0008	push buf
0009	push input
0010	goto strcpy
0011	return
0012	bar:
0013	push ”Gotcha!”
0014	goto printf
0015	pop
0016	return

Program.exe ”baz”

Stack

Addr	Data	
5601		
5602	5610	
1	5604	
2	b	buf
3	a	
4	z	
5607	0004	ret adr foo
5608	b	
1	a	
5610	z	

Addr	Code
0001	main:
0002	push argv[0]
0003	goto foo
0004	pop
0005	goto exit
0006	foo:
0007	allocate buf
0008	push buf
0009	push input
0010	goto strcpy
0011	return
0012	bar:
0013	push ”Gotcha!”
0014	goto printf
0015	pop
0016	return

Program.exe ”baz12”

Stack

Addr	Data	
5601		
5602	5610	
1	5604	
2	b	buf
3	a	
4	z	
5607	0012	ret adr foo
5608	b	
1	a	
5610	z	
5611	12	

Addr	Code
0001	main:
0002	push argv[0]
0003	goto foo
0004	pop
0005	goto exit
0006	foo:
0007	allocate buf
0008	push buf
0009	push input
0010	goto strcpy
0011	return
0012	bar:
0013	push ”Gotcha!”
0014	goto printf
0015	pop
0016	return

What was wrong?

- We copied into buf and did not check if we had room
- Values outside were changed=> program behavior changed!

Solution?

- Change Language :)
  - Not (always) an option :(
- Write better code!!!
  - Education
  - ”Secure” libraries

Buffer overflows: morale

- Attacks that directly target the Trusted Computing Base
- Serious!
  - Undermines most security policies
- Solution primarily to write robust code.