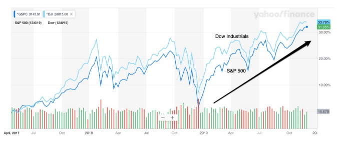
VERIFICATION OF SECURITY PROTOCOLS IN PROVERIF AND TAMARIN

Tom Chothia
University of Birmingham

Introduction

- These talks are about modelling and verifying security protocols.
- Modelling a protocol in Tamarin or ProVerif is a great way to understanding
- How this can be useful to you:
  - Analyzing a system: modelling in Tamarin or ProVerif is a great way to find bugs.
  - Proposing a protocol: formal models makes the design precise and increase confidence in its security.

1					7		9	
	3			2				8
		9	6			5		
		5	3			9		
	1			8				2
6					4			
3							1	
	4							7
		7				3		



Lecture 1: Ancient History

- Part 1: An information introduction to protocols
  - How the Internet works
  - Simple protocols in Alice and Bob notation.
  - Some protocol examples: NSP, Station-to-Station protocol, TLS, Wireguard.
  - Lots of security properties for protocols: security, authentication, forwards secrets, key impersonation security, post compromise security, …
- Part 2: Formal methods
  - λ-calculus, IMP, CCS,
  - pi-calculus: syntax, semantics and bisimulation
  - Case study: modelling an anonymous P2P system in pi-calculus

Lecture 2: Tamarin and ProVerif

- Part 1: ProVerif
  - The applied pi-calculus
  - The Proverif tool
  - Case study: traceability for e-passports
  - Case study: a protocols for pacemakers
- Part 2: Tamarin
  - The Tamarin Prover
  - How Tamarin works
  - Case study: Checking the WPA spec.

Lecture 3: EMV

- Part 1:
  - An overview of the EMV protocols.
  - Relay attacks
  - Distance bounding protections in applied pi-caluclus and Tamarin
- Part 2:
  - More EMV!
  - An attack against Apple Pay
  - Lots of formal models.

Lecture 1: Ancient History

- Part 1: An information introduction to protocols
  - How the Internet works
  - Simple protocols in Alice and Bob notation.
  - Some protocol examples: NSP, Station-to-Station protocol, TLS, Wireguard.
  - Lots of security properties for protocols: security, authentication, forwards secrets, key impersonation security, post compromise security, …

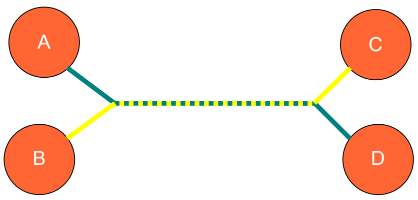
The Start 1969

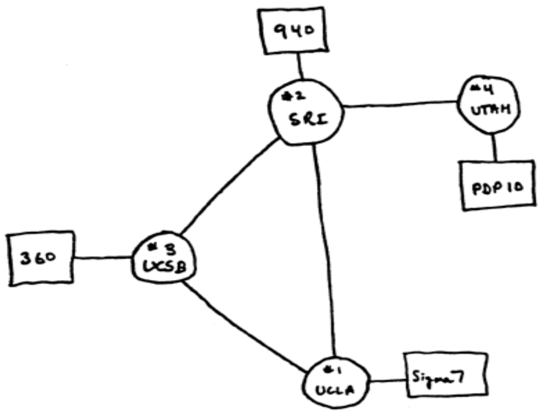
The US Defense Advanced Research Projects Agency (then ARPA now DARPA) gives research grants to universities to buy computers.

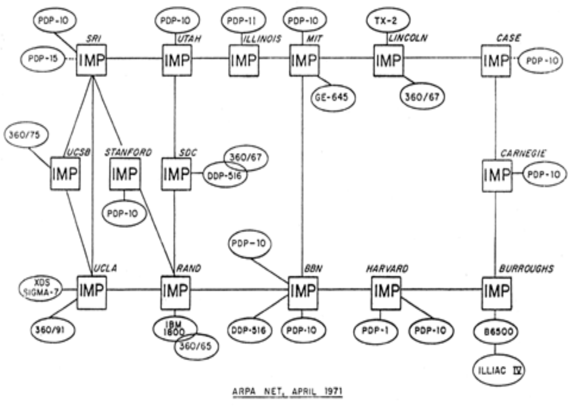
They decide to link their computers.

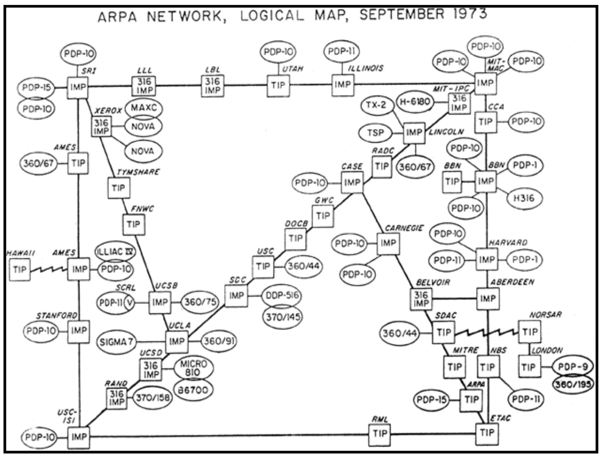
But how?

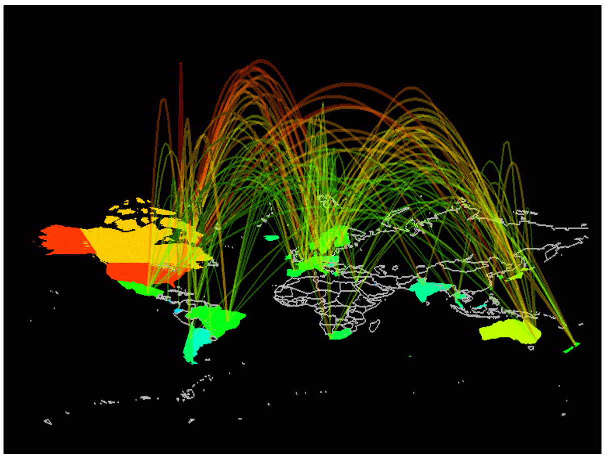
But if everyone just sends a small packet of data, they can both use the line at the same.

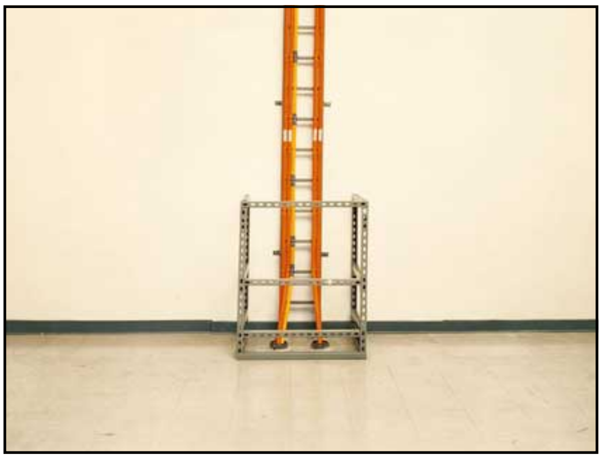




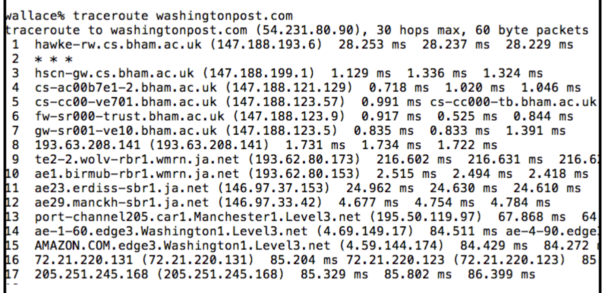




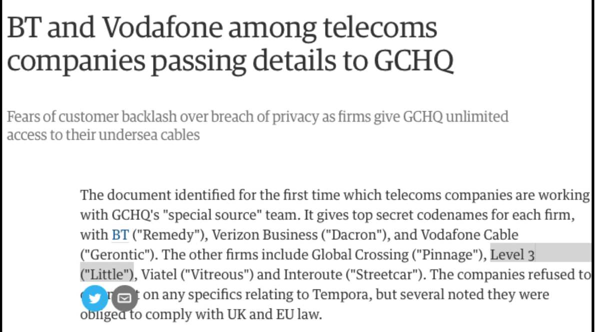




Birmingham to the Washington Post



```
wallace% traceroute washingtonpost.com
traceroute to washingtonpost.com (54.231.80.90), 30 hops max, 60 byte packets
 1  hawke-rw.cs.bham.ac.uk (147.188.193.6)  28.253 ms  28.237 ms  28.229 ms
 2  * * *
 3  hscn-gw.cs.bham.ac.uk (147.188.199.1)  1.129 ms  1.336 ms  1.324 ms
 4  cs-ac00b7e1-2.bham.ac.uk (147.188.121.129)  0.718 ms  1.020 ms  1.046 ms
 5  cs-cc00-ve701.bham.ac.uk (147.188.123.57)  0.991 ms cs-cc000-tb.bham.ac.uk
 6  fw-sr000-trust.bham.ac.uk (147.188.123.9)  0.917 ms  0.525 ms  0.844 ms
 7  gw-sr001-ve10.bham.ac.uk (147.188.123.5)  0.835 ms  0.833 ms  1.391 ms
 8  193.63.208.141 (193.63.208.141)  1.731 ms  1.734 ms  1.722 ms
 9  te2-2.wolv-rbr1.wmrn.ja.net (193.62.80.173)  216.602 ms  216.631 ms  216.6
10  ae1.birmub-rbr1.wmrn.ja.net (193.62.80.153)  2.515 ms  2.494 ms  2.418 ms
11  ae23.erdiss-sbr1.ja.net (146.97.37.153)  24.962 ms  24.630 ms  24.610 ms
12  ae29.manckh-sbr1.ja.net (146.97.33.42)  4.677 ms  4.754 ms  4.784 ms
13  port-channel205.car1.Manchester1.Level3.net (195.50.119.97)  67.868 ms  64
14  ae-1-60.edge3.Washington1.Level3.net (4.69.149.17)  84.511 ms ae-4-90.edge
15  AMAZON.COM.edge3.Washington1.Level3.net (4.59.144.174)  84.429 ms  84.272
16  72.21.220.131 (72.21.220.131)  85.204 ms 72.21.220.123 (72.21.220.123)  85
17  205.251.245.168 (205.251.245.168)  85.329 ms  85.802 ms  86.399 ms
```



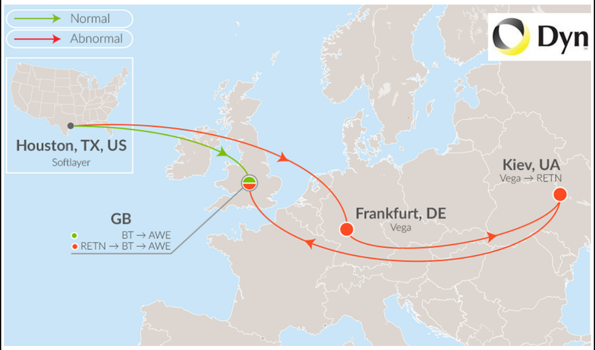
BT and Vodafone among telecoms companies passing details to GCHQ

Fears of customer backlash over breach of privacy as firms give GCHQ unlimited access to their undersea cables

The document identified for the first time which telecoms companies are working with GCHQ's "special source" team. It gives top secret codenames for each firm, with BT ("Remedy"), Verizon Business ("Dacron"), and Vodafone Cable ("Gerontic"). The other firms include Global Crossing ("Pinnage"), Level 3 ("Little"), Viatel ("Vitreous") and Interoute ("Streetcar"). The companies refused to [illegible] on any specifics relating to Tempora, but several noted they were obliged to comply with UK and EU law.

- http://www.theguardian.com/business/2013/aug/02/telecoms-bt-vodafone-cables-gchq

Redirected traffic to UK Atomic Weapons Establishment



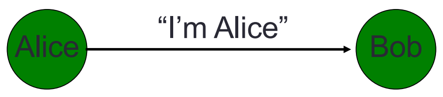
https://dyn.com/blog/uk-traffic-diverted-ukraine/

Dolev–Yao Attacker model

- Attack owns the network.
- They can intercept, replay, cut up and rearrange any messages.
- It's impossible to tell where a message came from. Attack can fake message.
- BUT the crypto is unbreakable and correct!

A Simple Protocol

“A” sends message “M” to “B”:



written as:

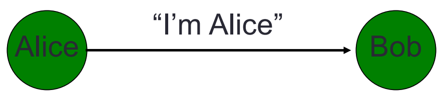
Alice → Bob : “I’m Alice”

Rules

We write down protocols as a list of messages sent between “principals”, e.g.

1. A → B : “Hello”
2. B → A : “Offer”
3. A → B : “Accept”

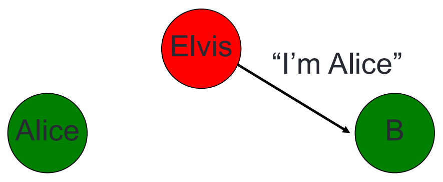
A Simple Protocol



A → B : “I’m Alice”

Message “I’m Alice” can be read by “The Attacker”.

A Simple Protocol

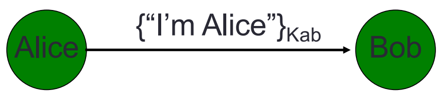


“The Attacker” can pretend to be anyone.

E(A) → B : “I’m Alice”

A Simple Protocol

$\{\_\}_{Kab}$ means symmetric key encryption



A → B : $\{\text{“I’m Alice”}\}_{Kab}$

If Alice and Bob share a key “Kab”,
then Alice can encrypt her message.

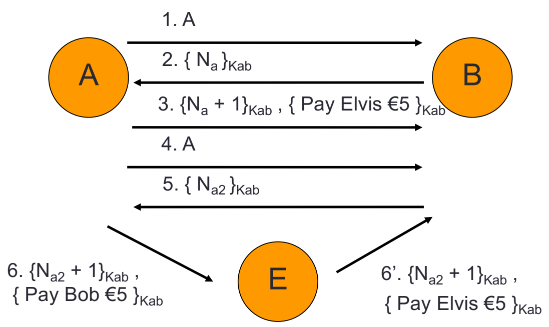
A Simple Protocol

A → E(B) : $\{\text{“I’m Alice”}\}_{Kab}$
E(A) → B : $\{\text{“I’m Alice”}\}_{Kab}$

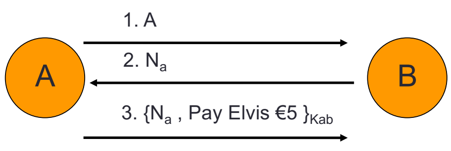
Attacker can intercept and replay messages.

Assume the attacker “owns” the network.

A Nonce


1. A
2. { N_a }$_{Kab}$
3. {N_a + 1}$_{Kab}$, { Pay Elvis €5 }$_{Kab}$
4. A
5. { N_{a2} }$_{Kab}$
6. {N_{a2} + 1}$_{Kab}$, { Pay Bob €5 }$_{Kab}$
6'. {N_{a2} + 1}$_{Kab}$, { Pay Elvis €5 }$_{Kab}$
A
B
E

A Better Protocol



1. A → B : A
2. B → A : N_a
3. A → B : {N_a, Pay Elvis €5 }$_{Kab}$

Key Establishment Protocol

This protocol was possible because A and B shared a key.

Often the principals need to set up a session key using a “Key Establishment Protocol”.

They must either know each others public keys or use a “Trusted Third Party” (TTP).

Protocol Goals

- Keep the **keys set up secret and ensure they are fresh.**

The Needham-Schroeder Public Key Protocol

Assume Alice and Bob know each others public keys, can they set up a symmetric key?

1. A → B : E_B(N_a, A)
2. B → A : E_A(N_a, N_b)
3. A → B : E_B(N_b)

E_X(_) means public key encryption

N_a and N_b can then be used to generate a symmetric key

An Attack Against the Needham-Schroeder Protocol

The attacker acts as a man-in-the-middle:

1. A → C : E_C(N_a, A)
    1`. C(A) → B : E_B(N_a, A)
    2`. B → C(A) : E_A(N_a, N_b)
2. C → A : E_A(N_a, N_b)
3. A → C : E_C(N_b)
    3`. C(A) → B : E_B(N_b)

The Corrected Version

A very simple fix:

1. $A \rightarrow B : E_B(N_a, A)$
2. $B \rightarrow A : E_A(N_a, N_b, B)$
3. $A \rightarrow B : E_B(N_b)$

Protocol Goals

- Keys set up are **secret and fresh**.
- **Authenticate: both parties**, i.e., both are talking to who they really believe they are talking to.

Forward Secrecy

$A \rightarrow B : E_B(N_a, A)$
$B \rightarrow A : E_A(N_a, N_b, B)$
$A \rightarrow B : E_B(N_b)$
$B \rightarrow A : \{M\}_{key(Na, Nb)}$

Secure against the "standard" attacker.
- intercept, replay, delete, alter.

- But what about the governments?
- After the protocol runs, governments can legally force people to hand over their private keys.
- Can we protect against this?

Forward Secrecy

- A protocol has "Forward Secrecy" if it keeps the message secret from an attacker who has:
  - A recording of the protocol run
  - The long-term keys of the principals.
- Protection against
  - a government that can force people to give up their keys,
  - or hackers that might steal them.

Protocol Goals

- Keys set up are **secret and fresh**.
- **Authenticate: both parties**, i.e., both are talking to who they really believe they are talking to.
- **Forward Secrecy.**

Diffie-Hellman

- Diffie-Hellman is a widely used key agreement protocol.
- It relies on some number theory:
  - $a \bmod b = n$ where for some "m" : $a = m.b + n$
- The protocol uses two public parameters
  - generator "g" (often 160 bits long)
  - prime "p" (often 1024 bits long)

Diffie-Hellman

- Alice and Bob pick random numbers r_A and r_B and find "$t_A = g^{r_A} \bmod p$" and "$t_B = g^{r_B} \bmod p$"
- The protocol just exchanges these numbers:
  1. $A \to B : t_A$
  2. $B \to A : t_B$
- "Alice" calculates "$t_B^{r_A} \bmod p$" and "Bob" "$t_A^{r_B} \bmod p$" this is the key:
  - $K = g^{r_A r_B} \bmod p$

Diffie-Hellman

1. $A \to B : (\overbrace{g \times g \times .. \times g}^{r^A \text{ times}}) \bmod p$
2. $B \to A : (\overbrace{g \times g \times \ldots. \times g}^{r^B \text{ times}}) \bmod p$

Alice calculates

$$\overbrace{(\overbrace{g \times\ldots\times g}^{r^B \text{ times}}) \bmod p \times (\overbrace{g \times\ldots\times g}^{r^B \text{ times}}) \bmod p \times \ldots \times (\overbrace{g \times\ldots\times g}^{r^B \text{ times}}) \bmod p}^{r^A \text{ times}}$$

$$= (\underbrace{g \times g \times g \ldots\times g}_{r^B r^A \text{ times}}) \bmod p$$

Diffie-Hellman

1. $A \to B : (\overbrace{g \times g \times .. \times g}^{r^A \text{ times}}) \bmod p$
2. $B \to A : (\overbrace{g \times g \times \ldots. \times g}^{r^B \text{ times}}) \bmod p$

Bob calculates

$$\overbrace{(\overbrace{g \times\ldots\times g}^{r^A \text{ times}}) \bmod p \times (\overbrace{g \times\ldots\times g}^{r^A \text{ times}}) \bmod p \times \ldots \times (\overbrace{g \times\ldots\times g}^{r^A \text{ times}}) \bmod p}^{r^B \text{ times}}$$

$$= (\underbrace{g \times g \times g \ldots\times g}_{r^A r^B \text{ times}}) \bmod p$$

Station-to-Station Protocol

1. $A \to B : g^x$
2. $B \to A : g^y, E_A(S_B(g^y,g^x))$
3. $A \to B : E_B(S_A(g^x,g^y))$
4. $B \to A : \{ M \}_{g^{xy}}$

$S_X(\_)$ means signed by X

- x,y, g^{xy} are not stored after the protocol run.
- A & B's keys don't let attacker read M
- STS has forward secrecy

A Protocol without Mutual Belief

1. $A \to B : g^x$
    - 1'. $B(A) \to C : g^x$
    - 2'. $C \to B(A) : g^y, E_A(S_C(g^y,g^x))$
2. $B \to A : g^y, E_A(S_B(g^y,g^x))$
3. $A \to B : E_B(S_A(g^y,g^x))$
    - 3'. $B(A) \to C: E_C(S_A(g^x,g^y))$
4. $A \to C : \{ M \}_{g^{xy}}$

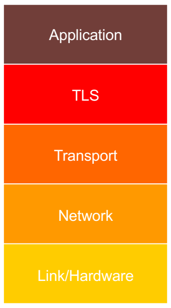
Alice thinks she is talking to Bob but is really talking to Charlie.

The Internet Protocol Stack with TLS

The TLS layer runs between the Application and Transport layer.

The encryption is transparent to the Application layer.

Normal TCP and IP protocols etc. can be used at the low layers



Transport Layer Security TLS 1.2

1. C →S : N_C
2. S →C : N_S , $Cert_S$
3. C →S : E_S(K_seed), $\{Hash1\}_{KCS}$
4. S →C : $\{Hash2\}_{KCS}$

Hash 1 = #(N_C,N_S, E_S(K_seed))
Hash 2 = #(N_C,N_S, E_S(K_seed), $\{Hash1\}_{KCS}$)

K_{CS} is a session key based key derivation function (kdf) on N_C, N_S & K_seed,
All data is then encrypted with K_{CS} and hashed for integrity.

Transport Layer Security (TLS)

1. C →S : N_C, Possible CipherSuites
2. S →C : N_S , $Cert_S$, CipherSuite
3. C →S : E_S(K_seed), $\{Hash1\}_{KCS}$
4. S →C : $\{Hash2\}_{KCS}$

Hash 1 = #(N_C,N_S, E_S(K_seed))
Hash 2 = #(N_C,N_S, E_S(K_seed), $\{Hash1\}_{KCS}$)

K_{CS} is a session key based key derivation function (kdf) on N_C, N_S & K_seed,

Protocol Goals

- Keys set up are **secret and fresh**.
- **Authenticate: both parties**, i.e., both are talking to who they really believe they are talking to.
- **Forward Secrecy.**
- **Downgrade Attacks**

Transport Layer Security (TLS)

1. C → S : **ClientHello*:*** N_C, Possible CipherSuites
2. S →C : N_S , $Cert_S$, CipherSuite
3. C →S : E_S(K_seed), $\{Hash1\}_{KCS}$
4. S →C : $\{Hash2\}_{KCS}$

Hash 1 = #(N_C,N_S, E_S(K_seed))
Hash 2 = #(N_C,N_S, E_S(K_seed), $\{Hash1\}_{KCS}$)

K_{CS} is a session key based key derivation function (kdf) on N_C, N_S & K_seed,

Transport Layer Security (TLS)

1. C → S : **ClientHello*:*** N_C, Possible CipherSuites
2. S → C : **ServerHello:** N_S, CipherSuite
2. S → C : **Certificate**: $Cert_S$
2. S → C : **ServerHelloDone**
3. C →S : E_S(K_seed), $\{Hash1\}_{KCS}$
4. S →C : $\{Hash2\}_{KCS}$

K_{CS} is a session key based key derivation function (kdf) on N_C, N_S & K_seed,

Transport Layer Security (TLS)

1. C → S : **ClientHello*:*** N_C, Possible CipherSuites
2. S → C : **ServerHello:** N_S, CipherSuite
2. S → C : **Certificate**: $Cert_S$
2. S → C : **ServerHelloDone**
3. C → S : **ClientKeyExchange** E_S(K_seed),
3. C → S : **ChangeCipherSpec**
3. C → S : $\{\text{Hash of previous message}\}_{KCS}$
4. S →C : $\{Hash2\}_{KCS}$

K_{CS} is a session key based key derivation function (kdf) on N_C, N_S & K_seed

Transport Layer Security (TLS)

1. C → S : **ClientHello***:* N_C, Possible CipherSuites
2. S → C : **ServerHello:** N_S , $Cert_S$, CipherSuite
2. S → C : **Certificate**: $Cert_S$
2. S → C : **ServerHelloDone**
3. C → S : **ClientKeyExchange:** $E_S(K_{seed})$
3. C → S : **ChangeCipherSpec**
3. C → S : $\{$Hash of previous message$\}_{KCS}$
4. S → C : **ChangeCipherSpec**
4. S → C : $\{$Hash of previous message$\}_{KCS}$

TLS-DHE

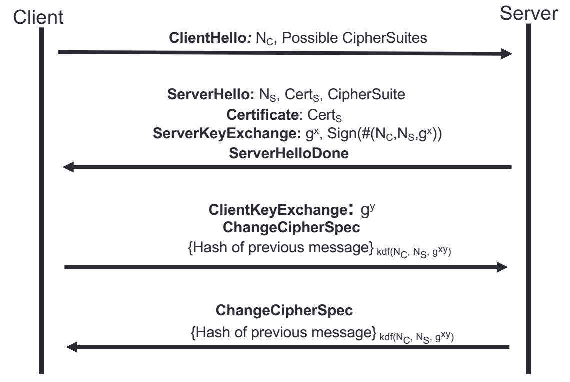
A variant uses Diffie-Hellman for "forward secrecy"

i.e., if someone gets the servers key later, they can't go back and break a recording of the traffic.

1. C → S : N_C , Possible CipherSuites
2. S → C : N_S , g^x, $Cert_S$, $Sign_S(\#(N_C,N_S,g^x))$, CipherSuite
3. C → S : g^y, $\{\#(\text{All previous messages})\}_{kdf(N_C,\ N_S,\ g^{xy})}$
4. S → C : $\{\#(\text{All previous messages})\}_{kdf(N_C,\ N_S,\ g^{xy})}$

Transport Layer Security (TLS)

1. C → S : **ClientHello***:* N_C, Possible CipherSuites
2. S → C : **ServerHello:** N_S , $Cert_S$, CipherSuite
2. S → C : **Certificate**: $Cert_S$
2. S → C : **ServerKeyExchange:** g^x, $Sign(\#(N_C,N_S,g^x))$
2. S → C : **ServerHelloDone**
3. C → S : **ClientKeyExchange:** g^y
3. C → S : **ChangeCipherSpec**
3. C → S : $\{$Hash of previous message$\}_{kdf(N_C,\ N_S,\ g^{xy})}$
4. S → C : **ChangeCipherSpec**
4. S → C : $\{$Hash of previous message$\}_{kdf(N_C,\ N_S,\ g^{xy})}$



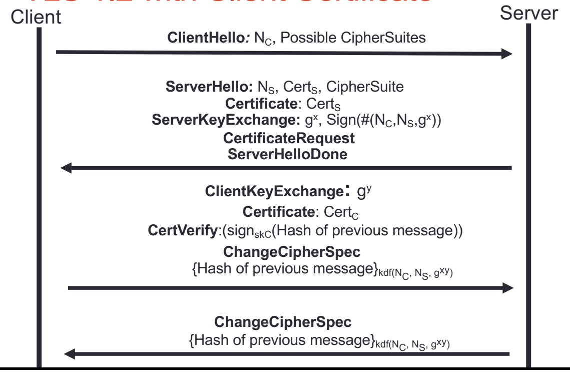
ClientHello*:* N_C, Possible CipherSuites

ServerHello: N_S, $Cert_S$, CipherSuite
Certificate: $Cert_S$
ServerKeyExchange: g^x, $Sign(\#(N_C,N_S,g^x))$
ServerHelloDone

ClientKeyExchange: g^y
ChangeCipherSpec
$\{$Hash of previous message$\}_{kdf(N_C,\ N_S,\ g^{xy})}$

ChangeCipherSpec
$\{$Hash of previous message$\}_{kdf(N_C,\ N_S,\ g^{xy})}$

TLS 1.2 with Client Certificate



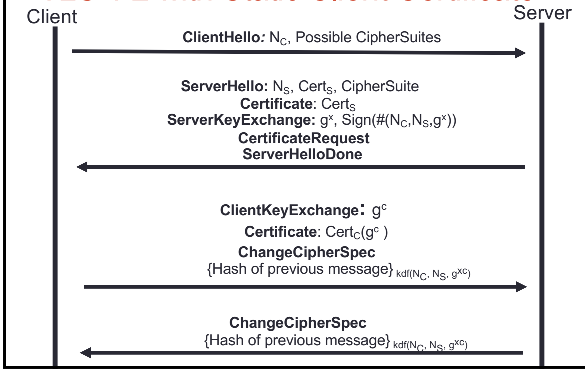
ClientHello*:* N_C, Possible CipherSuites

ServerHello: N_S, $Cert_S$, CipherSuite
Certificate: $Cert_S$
ServerKeyExchange: g^x, $Sign(\#(N_C,N_S,g^x))$
CertificateRequest
ServerHelloDone

ClientKeyExchange: g^y
Certificate: $Cert_C$
CertVerify:($sign_{skC}$(Hash of previous message))
ChangeCipherSpec
$\{$Hash of previous message$\}_{kdf(N_C,\ N_S,\ g^{xy})}$

ChangeCipherSpec
$\{$Hash of previous message$\}_{kdf(N_C,\ N_S,\ g^{xy})}$

TLS 1.2 with Static Client Certificate



ClientHello*:* N_C, Possible CipherSuites

ServerHello: N_S, $Cert_S$, CipherSuite
Certificate: $Cert_S$
ServerKeyExchange: g^x, $Sign(\#(N_C,N_S,g^x))$
CertificateRequest
ServerHelloDone

ClientKeyExchange: g^c
Certificate: $Cert_C(g^c)$
ChangeCipherSpec
$\{$Hash of previous message$\}_{kdf(N_C,\ N_S,\ g^{xc})}$

ChangeCipherSpec
$\{$Hash of previous message$\}_{kdf(N_C,\ N_S,\ g^{xc})}$

TLS 1.2 is a mess

- There are too many cipher suites.
- There are too many options.
- Due to this there have been lots of attacks against particular cipher suites, or rare options.
  - E.g. https://en.wikipedia.org/wiki/Transport_Layer_Security_Security
- You need 2 round trips before you can start sending data.
- Many attacks based on mistakes in implementations.
  - The complexity makes this more likely.
- TLS 1.3 fixes this.

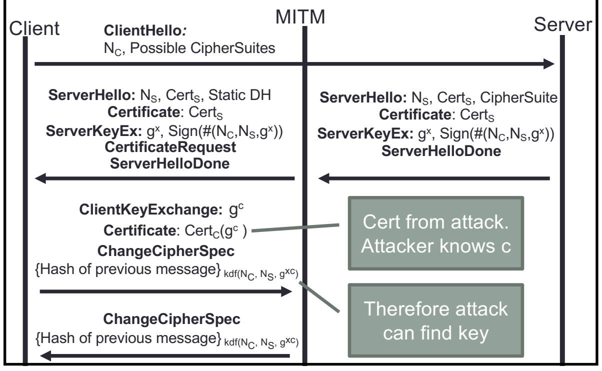
Key Compromise Impersonation Attack

- In a KCI attack: attack can control client's key.
- The attack is successful if the attack can pretend to be any server.
- E.g. your employer says you must install a client certificate to connect to your place of work and then uses this to MITM any connection.
- This is an edge case attack: if attack can install a certificate in the trust store then TLS is trivially broken.
- TLS with static DH and Client Certificate is vulnerable to a KCI attack.

Protocol Goals

- Keys set up are **secret and fresh**.
- **Authenticate: both parties**, i.e., both are talking to who they really believe they are talking to.
- **Forward Secrecy.**
- **Downgrade Attacks**
- **Key Compromise Impersonation Attack.**

TLS 1.2 KCI vs Static Client Certificate


Client
MITM
Server
ClientHello: N_C, Possible CipherSuites
ServerHello: N_S, Cert_S, Static DH
Certificate: Cert_S
ServerKeyEx: g^x, Sign(#(N_C,N_S,g^x))
CertificateRequest
ServerHelloDone
ServerHello: N_S, Cert_S, CipherSuite
Certificate: Cert_S
ServerKeyEx: g^x, Sign(#(N_C,N_S,g^x))
ServerHelloDone
ClientKeyExchange: g^c
Certificate: Cert_C(g^c)
ChangeCipherSpec
{Hash of previous message} kdf(N_C, N_S, g^xc)
ChangeCipherSpec
{Hash of previous message} kdf(N_C, N_S, g^xc)
Cert from attack. Attacker knows c
Therefore attack can find key

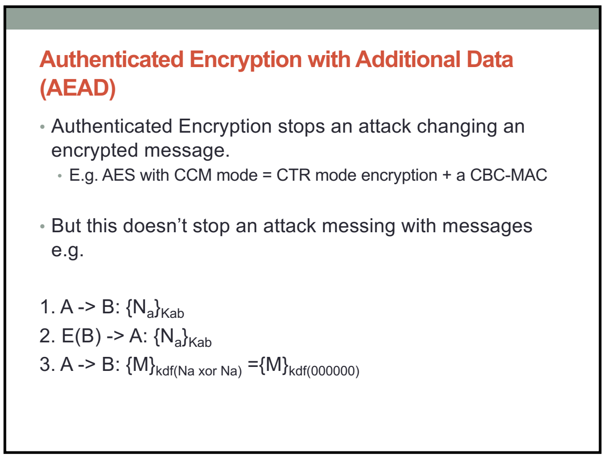
Protocol Goals

- Keys set up are **secret and fresh**.
- **Authenticate: both parties**, i.e., both are talking to who they really believe they are talking to.
- **Forward Secrecy.**
- **No weak cipher.**
- **Key Compromise Impersonation Attack.**
- **Protocol should not be complex or have lots of different modes.**

Authenticated Encryption with Additional Data (AEAD)

- Authenticated Encryption stops an attack changing an encrypted message.
  - E.g. AES with CCM mode = CTR mode encryption + a CBC-MAC
- But this doesn't stop an attack messing with messages e.g.

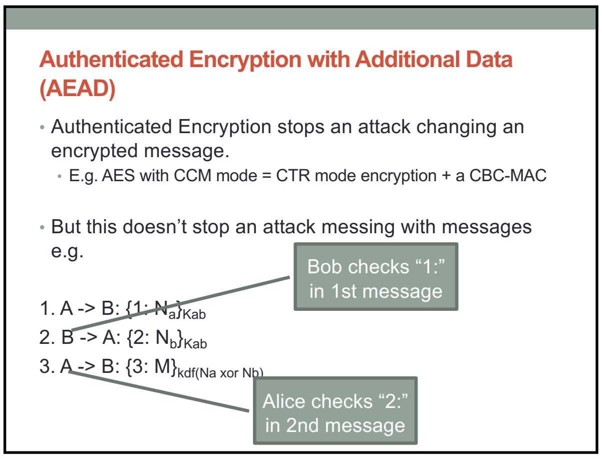
1. A -> B: $\{N_a\}_{Kab}$
2. B -> A: $\{N_b\}_{Kab}$
3. A -> B: $\{M\}_{kdf(Na\ xor\ Nb)}$



Authenticated Encryption with Additional Data (AEAD)

- Authenticated Encryption stops an attack changing an encrypted message.
  - E.g. AES with CCM mode = CTR mode encryption + a CBC-MAC
- But this doesn't stop an attack messing with messages e.g.

1. A -> B: $\{N_a\}_{Kab}$
2. E(B) -> A: $\{N_a\}_{Kab}$
3. A -> B: $\{M\}_{kdf(Na\ xor\ Na)}$ $=\{M\}_{kdf(000000)}$



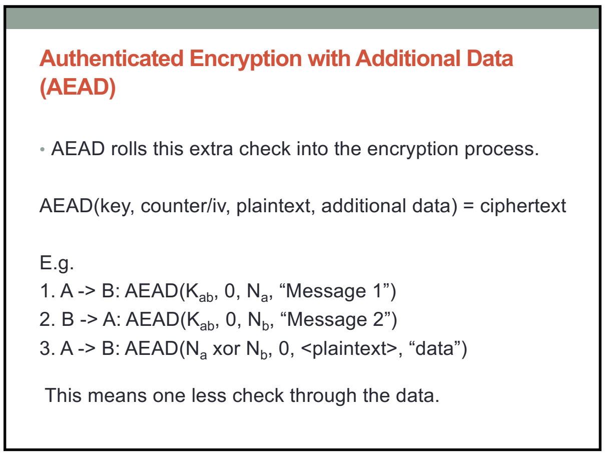
Authenticated Encryption with Additional Data (AEAD)

- Authenticated Encryption stops an attack changing an encrypted message.
  - E.g. AES with CCM mode = CTR mode encryption + a CBC-MAC
- But this doesn't stop an attack messing with messages e.g.

1. A -> B: $\{1: N_a\}_{Kab}$
2. B -> A: $\{2: N_b\}_{Kab}$
3. A -> B: $\{3: M\}_{kdf(Na\ xor\ Nb)}$

Bob checks "1:" in 1st message

Alice checks "2:" in 2nd message



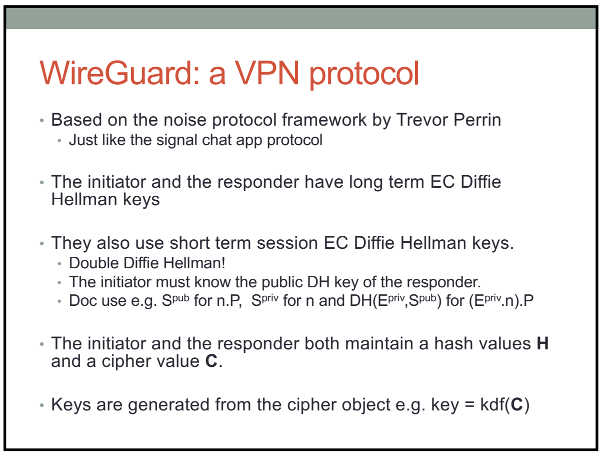
Authenticated Encryption with Additional Data (AEAD)

- AEAD rolls this extra check into the encryption process.

AEAD(key, counter/iv, plaintext, additional data) = ciphertext

E.g.
1. A -> B: AEAD(K_{ab}, 0, N_a, "Message 1")
2. B -> A: AEAD(K_{ab}, 0, N_b, "Message 2")
3. A -> B: AEAD(N_a xor N_b, 0, <plaintext>, "data")

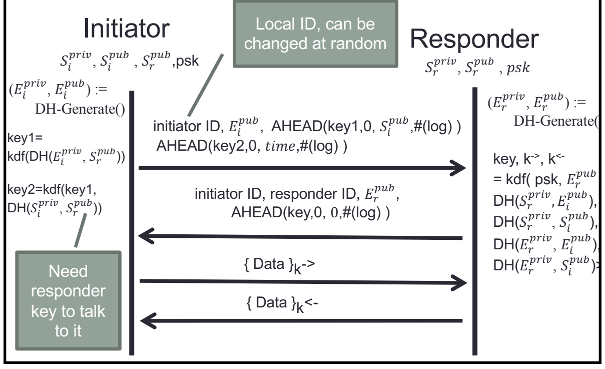
This means one less check through the data.



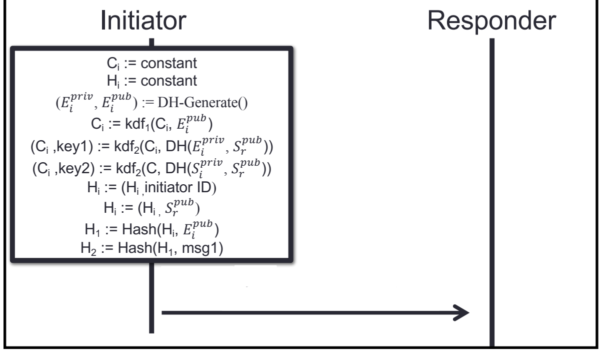
WireGuard: a VPN protocol

- Based on the noise protocol framework by Trevor Perrin
  - Just like the signal chat app protocol
- The initiator and the responder have long term EC Diffie Hellman keys
- They also use short term session EC Diffie Hellman keys.
  - Double Diffie Hellman!
  - The initiator must know the public DH key of the responder.
  - Doc use e.g. S^{pub} for n.P, S^{priv} for n and DH(E^{priv},S^{pub}) for (E^{priv}.n).P
- The initiator and the responder both maintain a hash values **H** and a cipher value **C**.
- Keys are generated from the cipher object e.g. key = kdf(**C**)

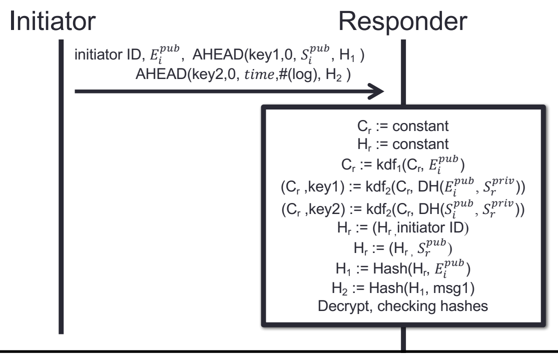
WireGuard


Initiator
$S_i^{priv}, S_i^{pub}, S_r^{pub}$,psk
Local ID, can be changed at random
Responder
$S_r^{priv}, S_r^{pub}, psk$
(E_i^{priv}, E_i^{pub}) := DH-Generate()
key1= kdf(DH(E_i^{priv}, S_r^{pub}))
key2=kdf(key1, DH(S_i^{priv}, S_r^{pub}))
Need responder key to talk to it
initiator ID, E_i^{pub}, AHEAD(key1,0, S_i^{pub},#(log))
AHEAD(key2,0, $time$,#(log))
initiator ID, responder ID, E_r^{pub},
AHEAD(key,0, 0,#(log))
{ Data }$_{k^{->}}$
{ Data }$_{k^{<-}}$
(E_r^{priv}, E_r^{pub}) := DH-Generate(
key, $k^{->}$, $k^{<-}$
= kdf(psk, E_r^{pub}
DH(S_r^{priv}, E_i^{pub}),
DH(S_r^{priv}, S_i^{pub}),
DH(E_r^{priv}, E_i^{pub})
DH(E_r^{priv}, S_i^{pub})

Running Cipher and Hash Values


Initiator
Responder
C_i := constant
H_i := constant
(E_i^{priv}, E_i^{pub}) := DH-Generate()
C_i := kdf$_1$(C_i, E_i^{pub})
(C_i ,key1) := kdf$_2$(C_i, DH(E_i^{priv}, S_r^{pub}))
(C_i ,key2) := kdf$_2$(C, DH(S_i^{priv}, S_r^{pub}))
H_i := (H_i ,initiator ID)
H_i := (H_i , S_r^{pub})
H_1 := Hash(H_i, E_i^{pub})
H_2 := Hash(H_1, msg1)

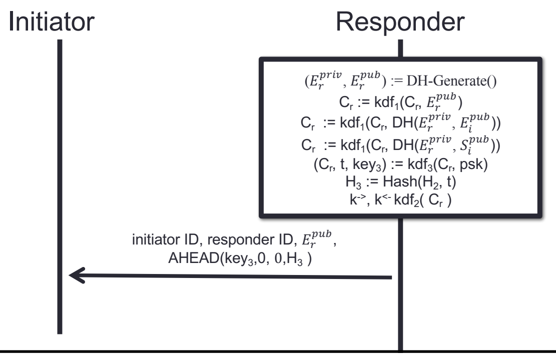
Running Cipher and Hash Values



initiator ID, E_i^{pub}, AHEAD(key1,0, S_i^{pub}, H_1)
AHEAD(key2,0, $time$,#(log), H_2)

C_r := constant
H_r := constant
C_r := kdf_1(C_r, E_i^{pub})
(C_r ,key1) := kdf_2(C_r, DH(E_i^{pub}, S_r^{priv}))
(C_r ,key2) := kdf_2(C_r, DH(S_i^{pub}, S_r^{priv}))
H_r := (H_r ,initiator ID)
H_r := (H_r , S_r^{pub})
H_1 := Hash(H_r, E_i^{pub})
H_2 := Hash(H_1, msg1)
Decrypt, checking hashes

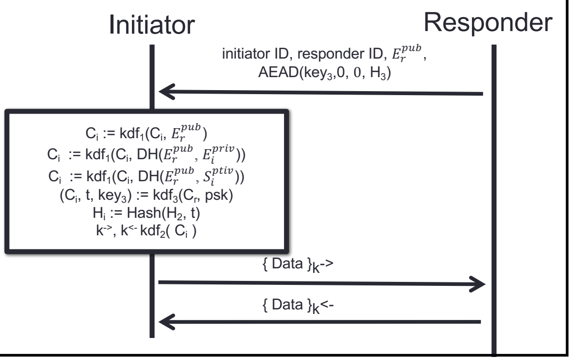
Running Cipher and Hash Values



(E_r^{priv}, E_r^{pub}) := DH-Generate()
C_r := kdf_1(C_r, E_r^{pub})
C_r := kdf_1(C_r, DH(E_r^{priv}, E_i^{pub}))
C_r := kdf_1(C_r, DH(E_r^{priv}, S_i^{pub}))
(C_r, t, key_3) := kdf_3(C_r, psk)
H_3 := Hash(H_2, t)
$k^{->}$, $k^{<-}$ kdf_2(C_r)

initiator ID, responder ID, E_r^{pub},
AHEAD(key_3,0, 0,H_3)

Running Cipher and Hash Values



initiator ID, responder ID, E_r^{pub},
AEAD(key_3,0, 0, H_3)

C_i := kdf_1(C_i, E_r^{pub})
C_i := kdf_1(C_i, DH(E_r^{pub}, E_i^{priv}))
C_i := kdf_1(C_i, DH(E_r^{pub}, S_i^{ptiv}))
(C_i, t, key_3) := kdf_3(C_r, psk)
H_i := Hash(H_2, t)
$k^{->}$, $k^{<-}$ kdf_2(C_i)

{ Data }$_{k->}$

{ Data }$_{k<-}$

Rekeying

After 2mins or 2^{60} messages the handshake runs again

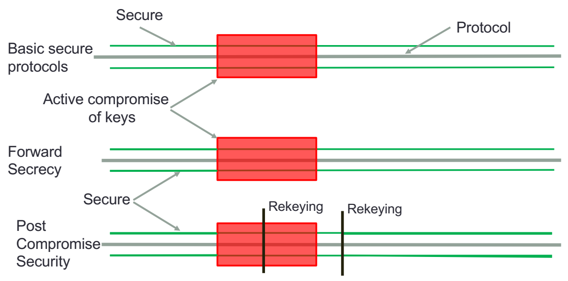
- New ephemeral DH keys
- Cipher and hash values not reset.

Symbol	Value
Rekey-After-Messages	2^{60} messages
Reject-After-Messages	$2^{64} - 2^{13} - 1$ messages
Rekey-After-Time	120 seconds
Reject-After-Time	180 seconds
Rekey-Attempt-Time	90 seconds
Rekey-Timeout	5 seconds
Keepalive-Timeout	10 seconds

- If the attacker gets key, they will loose access after rekeying, unless they actively MITM the handshake.
- Also stops attack getting a large amount of traffic encrypted with the same key.
  - Stops side channel attacks
  - Or yet to be discovered crypto attacks

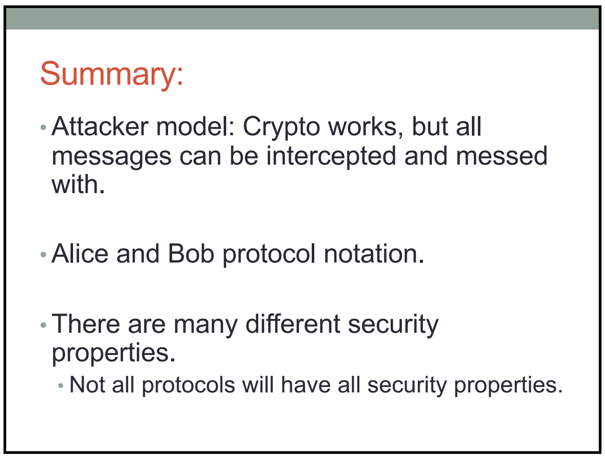
Post Compromise Security

- If attacker complete compromises a system, then stops post compromise security will restore the protection



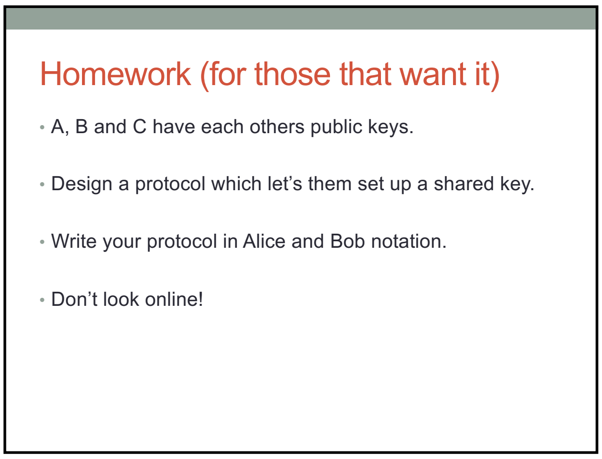
Protocol Goals

- Keys set up are **secret and fresh**.
- **Authenticate: both parties**, i.e., both are talking to who they really believe they are talking to.
- **Forward Secrecy.**
- **Downgrade Attacks**
- **Key Compromise Impersonation Attack.**
- **Protocol should not be complex or have lots of different modes.**
- **Post compromise security.**
- … … …



Summary:

- Attacker model: Crypto works, but all messages can be intercepted and messed with.
- Alice and Bob protocol notation.
- There are many different security properties.
  - Not all protocols will have all security properties.



Homework (for those that want it)

- A, B and C have each others public keys.
- Design a protocol which let's them set up a shared key.
- Write your protocol in Alice and Bob notation.
- Don't look online!