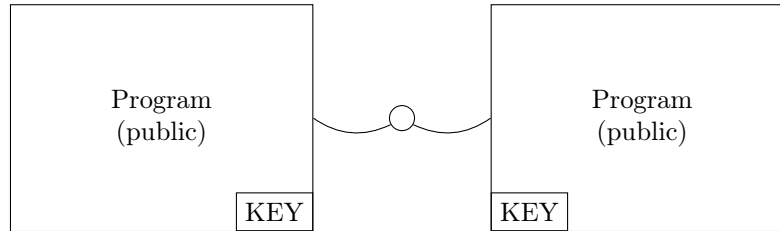


When we send data over an untrusted network, we need to encrypt the data! We do this according to *Kerchoff's Design Principal* := minimize what needs to be kept secret.



The question of security is a question of trust. Most programs run by the OS are not trusted (cat, grep, sh, gcc, make...) but a few are (login, sudo, su...), and can therefore call `setuid()`

If we are worried about security, the latter are the risk: how do we choose who to trust?

sudo	everything else
login	
kernel	
hw	

We call the set of all trusted programs the *trusted computing base*. We seek to minimize this.

BUT can we really trust anything? Take Ken Thompson, creator of Unix. He once wrote a Turing Award winning paper called "Reflections on Trusting Trust". The paper described a bug he built into Unix.

```
// in the login program, Ken added the following code
if(strcmp(user,"ken")==0) uid = 0;
// however, this would be easily found out, so in gcc, he adds:
if(compiling(login.c)) generate strcmp
// now no one looking at the login source code can see the bug
// he also adds in gcc
if(compiling(gcc.c)) generate generate strcmp
// he compiles gcc and edits the code back — the code is now gone!
```

Couldn't this be found in the assembly? NO:

- he can modify the code for `objdump`
- he can even make it so that using other code injects bugs!

The moral of the story is, "You need to trust your tools" (as an aside, it also shows gcc really should be in the TCB).

With this principal in mind, we model secure message passing:

$$A \rightarrow B \text{"I'm Alice"}^K$$

This is BAD; it is subject to replay attacks. Thus we need a *nonce* := a nonsense string used to verify possession of a key:

$$\begin{aligned} A &\rightarrow B \text{"I'm Alice"} \\ B &\rightarrow A \text{"<nonce>"} \\ A &\rightarrow B \{\text{"<nonce>"}\} \\ B &\rightarrow A \text{"OK"} \end{aligned}$$

We can now effectively verify identity, so how do we use this to send messages?

- we use a Hashed Message Authentication Code

$$ABBM \parallel \text{HMAC}(M, K)$$

how do we find an HMAC? We call our cryptographic friends for a *(Cryptographically) Secure Checksum Algorithm*, defined st if $\text{SHA}(M) = N$, it is hard to find M' st $\text{SHA}(M') = N$. We then let $\text{HMAC}(M, K) = f(\text{SHA}(K \parallel M))$

Now we are 100% secure, right?

- Of course not: SHA's can be broken — we are already on SHA 2.something

Distributed security can also be done with a virtual network inside of one Computer:

Typical authentication via ssh is multiphase :

1. establish connection with public/private key encryption
2. use the encrypted nonce as a shared secret key

Shared secrets are the model we used above; the public/private model is as such:

1. a message $A \rightarrow B$ is encrypted with B's public key
2. person A can then decrypt the message with their private key

So it is better to use this first, since it is slower but more secure. note: verifying a public key is not 100% exact.

This model is utilized by the *RSA encryption algorithm*

- The keys for the RSA algorithm are generated in the following way:
 1. Choose two distinct prime numbers p and q .
 - p & q should be chosen at random
 - p & q should be similar in magnitude but differ in length by a few digits
 - p and q are kept secret
 2. Compute $n = pq$.
 - n is used as the modulus for both keys
 - the length of n expressed in bits, is the key length
 3. Compute $\Phi(n) = (p - 1)(q - 1)$
 4. Choose an integer e such that $1 < e < \Phi(n)$ and e and $\Phi(n)$ share no factors besides 0.
 - an e with a short bit-length results in more efficiency
 - the most commonly chosen value for e is $2^{16} + 1 = 65,537$
 - the smallest (and fastest) possible value for e is 3
 5. Determine $d \equiv e^{-1} \pmod{\Phi(n)}$
- The *public key* consists of the modulus n and the public (or encryption) exponent e .
- The *private key* consists of the private (or decryption) exponent d .
 - p , q , and $\lambda(n)$ must also be kept secret because they can be used to calculate d .
 - They can all be discarded after d has been computed.

For example:

Suppose $P = 53$ and $Q = 59$.

Then $n = PQ = 3127$.

Let $e = 3$.

Then, $\Phi(n) = 3016$.

For $k = 2$, value of d is 2011.

We need to calculate $\Phi(n)$ such that $\Phi(n) = (P - 1)(Q - 1)$.

Then $d = (k * \Phi(n) + 1)/e$ for some integer k .

Thus for $k = 2$, value of d is 2011.