Can we do better than shared key cryptography?

Reminder: “shared key” cryptography rests on two (or more) parties each knowing a *secret* shared key. Diffie-Hellman is one way for two parties to generate such a key in the public eye.

Goal: get rid of the sharing of secret keys. We will still have secret keys, but they will be owned by individuals, not by groups.

Let’s follow the problem through the years, looking at the mathematicians who led us to an answer.
We would like a two functions such that, if given a message M we want to send, the first function will encrypt M and the second will decrypt it:

\[ M \rightarrow f_1(M) \rightarrow C \rightarrow f_2(C) \rightarrow M \]

The function \( f_1 \) is used by Bob and the function \( f_2 \) is used by Alice. The encrypted form of M, called C, can be sent in the clear.

Here is one example:

\[
\begin{align*}
    f_1(x) &= x \times k \\
    f_2(x) &= \frac{x}{k}
\end{align*}
\]

So Bob takes his message M and multiplies it by k (a public value) to get C. Alice takes C and divides by k to get back M.

What this rests on is that Eve cannot guess the two inverse functions. In some ways, it is still a shared key system, but now Bob and Alice are sharing their functions (keeping them secret).
Here is another attempt (that will succeed)

As before, we want this to hold:

\[ M \rightarrow f_1(M) \rightarrow C \rightarrow f_2(C) \rightarrow M \]

However, we will modify the problem as follows:

\[ M \rightarrow f_1(M,e) \rightarrow C \rightarrow f_2(C,d) \rightarrow M \]

So Bob takes his message M and uses some function f1 to encrypt. But now f1 also uses a public number e. Alice takes C and uses f2 to get back M. But f2 now takes a private number d, known only to Alice.

So the general idea is that Alice has two keys: a public one, e, that everyone uses to encrypt, and a private one, d, that only Alice knows.

Note that everyone knows the functions f1 and f2 in this scheme. The only secret is the number d.
Ok, how do we get these functions and numbers?

The Theorem Jockies

Fermat (pronounced FAIR MAH) proved the following theorem:

\[ m^{(p-1)} \mod p = 1, \text{ where } p \text{ is prime.} \]

This lead to a further theorem by Euler (pronounced OILER):

\[ m^{((p-1)*(q-1))} \mod n = 1, \text{ where } p \text{ and } q \text{ are prime, and } p*q=n. \]

Let me tweak this a bit to get us ready for the flashy step. I'll multiply both sides of Euler's equation by \( m \) and then do some simplification:

\[ m*(m^{((p-1)*(q-1))}) \mod n = m. \]
\[ (m^{*1})*(m^{((p-1)*(q-1))}) \mod n = m. \]
\[ m^{((p-1)*(q-1)+1)} \mod n = m. \]
I think we are close to having what we want:

\[ m^{((p-1)*(q-1)+1)} \mod n = m. \]

What we have is an operation, modular exponentiation, and a number, \(((p-1)*(q-1)+1)\), that will take \( m \) and produce \( m \). That's a good start, but not quite what we need.

What if we could somehow divide up the number into two parts, one that encrypts and one that decrypts. What we need is something like this:

\[ e*d=((p-1)*(q-1)+1). \]

If we could find these two numbers \( e \) and \( d \), then we know the following:

\[ m^{*(e*d)} \mod n = m. \]

Can you see where this is leading?
The Birth of PPK

Given:
\[ m^{(e \cdot d)} \mod n = m. \]

we know:
\[ m^{(e)} \mod n = c. \]
\[ c^{(d)} \mod n = m. \]

There it is. We've got it. Knowing \( e \) and \( n \), Bob can encrypt \( m \) as \( c \). Knowing \( d \) and \( n \), Alice can decrypt \( c \) as \( m \).

And now we make \( e, n \) public! Everyone knows them. So now Alice has two keys:

- \( e, n \) which are public
- \( d \) which is private

We call this PPK cryptography for Public-Private Key
The RSA Algorithm

I was a bit glib in producing e and d from the equation:

\[ e \cdot d = ((p-1) \cdot (q-1) + 1). \]

Solving this equation will take a bit more machinery. Let's use some example values to ground this. Assume the following (by convention, \( (p-1) \cdot (q-1) \) is called \( \phi \) - FEE.)

\[
\begin{align*}
    p &= 5 \\
    q &= 11 \\
    n &= 55 \\
    \phi &= (p-1) \cdot (q-1) = 40 \\
    \phi + 1 &= 41
\end{align*}
\]

That leaves us with the following equation to solve:

\[ e \cdot d = (\phi + 1) = 41. \]

Suppose I choose the value 7 for e - this will be the public key. I now have:

\[ 7 \cdot d = (\phi + 1) = 41. \]

All I need to do now is find the secret key, d, that solves the equation and I am ready to start encrypting and decrypting. But there is a bit of a snag: 41 is a prime and has no factors. There are no integer values for e and d that will solve the equation.

Dang!
Euclid to the rescue

Given:

$$7d = (\phi + 1) = 41.$$ 

I can translate 41 into 1 mod 40, and then come up with a new equation to solve:

$$7d = 1 \mod 40.$$ 

You will see an algorithm in Discrete Mathematics that will help you get a value for \(d\) (and then \(e\)):

Extended_Euclid(a, b):

if \(b = 0\) then return \((a, 1, 0)\)

\((d', x', y') \leftarrow \text{Extended_Euclid}(b, a \mod b)\)

\((d, x, y) \leftarrow (d', y', (x' - (a/b) \times y'))\)

return \((d, x, y)\)

Let’s just assume for now that we can set up all the numbers we need: \(n, e, d\).
Why is this hard to crack?

What does Eve know?
She knows the value of $e$ and $n$.
She knows that $n=p\times q$ and $p,q$ are both primes.
If she can find $p$ (or $q$), she can crack the whole thing.

Example to ground this. Recently, there was a worldwide effort to crack an $n$ of 129 digits. Computers from around the globe worked on it.

It took 8 months.

Today's algorithms use 150 digits numbers for $n$.

All this works because of two things:
1. There is no known alg for systematically generating primes, other than brute force.
2. It is not hard to find a random prime (to generate $n$).

Kind of like cracks in the street: easy to find a random one; hard to find a specific one.
In Summary

Alice generates five numbers: \( p, q, n, e, d \). She makes two of them public: \( n, e \). When someone (say, Bob) wants to send her something, he uses the following:

\[ m^e \mod n \rightarrow c \]

He then sends her \( c \). Alice gets \( c \) and uses the following:

\[ c^d \mod n \rightarrow m \]

One other trick. What if Alice sends Bob the following:

\[ m^d \mod n \rightarrow c \]

And Bob does the following:

\[ c^e \mod n \rightarrow m \]

Of course, Eve can do the same given \( e, n \) are known.

The point is that only Alice could have sent this message. It is a form of digital signature!