Digital Signatures

Welcome back. In this video, we're going to take a look at digital signatures. Digital signatures give us a digital equivalent to a handwritten signature, but they're even better. Where a handwritten signature can be easily forged, a well-designed and implemented digital signature cannot. A digital signature gives us some assurance in terms of the integrity of the message, as well as some origin authentication. So let's break it down. Let's have a high-level look at how digital signatures work in general. We have a private key. And we also have a public key. The private key is used for signing.

So we would refer to that using s. And the public key is used for verification. We'll refer to that as v. We have a message that we would like to send. And we want to prove that it's us that wrote it, and that it hasn't been messed with at all. So it hasn't been modified in any way by an attacker, for example. What we can do is we can take this message and apply our private key to encrypt that message. We can also send this along with the message itself.

So note, in particular, that this is not achieving the confidentiality of the message, but it is going to help us to prove that the message was written by me. And it's also going to help us prove that the message hasn't been altered. The recipient then receives these two pieces of information. And what they can do is they can take my public key and apply that to the message and then compare that to the message that we have here.

So if these two values are the same, then we can say, yes, everything's OK. If they are not the same, then something has gone wrong. And that value has been tampered with. Now, one of the things to note here is that if that message is especially small or especially large that can be difficult in a number of different ways. It can either impact the security. So, for example, if the message was just the number one, then you're not really going to be able to encrypt that in any meaningful way.

So instead, what we do is we make use of cryptographic hash functions. A cryptographic hash function allows us to take any length input and provides a fixed length output. So the common types of length that we see are lengths of 128-bit to 256-bit and so on and so forth. So it doesn't matter what the length of the original message is. The final output is a fixed length, which is determined by the algorithm that's used. Hashes have a number of properties.

The first is that of being determinant. That is the same message will always result in the same output. So if we think of the set of all messages compared to the set of all outputs or hash values, then we want a specific message here to go to a specific message there. And it shouldn't ever the same message go to multiple different outputs. For it to be cryptographically secure, we need to be sure that we're not going to have lots of different messages which could result in the same final output.

That is referred to as collision resistance, in that multiple values are unlikely to end with the same output. And the other thing that we're looking for is that it is really difficult to go backwards. So if we have the output value or hash value, it must be computationally infeasible to then determine what the original input was. This is referred to as the one-way property. So moving back to our example here of the digital signature, what we can do is rather than having that message, which could be problematic if it's too big because then you're sending a lot of information. It could take a while for that to happen.

Or if it's too small, in which case, it's unrealistic to expect meaningful encryption. And we can take that message and put it through a cryptographically secure hash function to get out our output, which can be referred to as our message digest. So we've now looked at a high-level as to how a digital signature signing and verification operation happens. But how does this actually help us? Well, the public key, which is linked to the private key used for signing means that that is the only public key which is going to be able to verify that signature.

As a result, this gives us some confidence that the person who sent it is indeed who they are claiming to be, as the public key matches the private key. We also have the addition of integrity. By making use of the comparison between the signed version, which has had the encryption taken off of it, versus the version which is sent alongside, we can see that there has been no change to the message itself. If an attacker were to try to change those values, or insert their own document to compare to the signature, the result of the verifying operation would be negative.

So where does this leave us in terms of real world applications? This is used particularly within authenticated key exchanges. We'll explore this in more depth elsewhere, but for the moment, consider the situation where instead Alice knows Bob's verifying key. So the key generation has happened for the digital signature side of things, and Alice already has that. If Eve were to intercept any communication of a key from Bob, if that key is signed, then they are not going to be able to change that out.

Alice is going to know that some form of man in the middle attack has been attempted. Note, however, that this is only authentication on one side. If we wanted to ensure authentication for both sides, we would need to do that for both Alice and Bob. And that would be referred to as mutual authentication, which we'll look at elsewhere. Now, you may realistically be saying, well, haven't we just moved the problem further down the line? What if we haven't agreed what Bob's verifying key is in advance? Well, again, this comes back to the real world application of this through digital certificates, which we'll explore elsewhere. But for now, it's suffice to say that this does have a real world application and is most likely being used at the moment whilst you are watching this video.

With all of our cryptographic protocols and primitives, there are associated standards. And this is no different for digital signatures. When you look to apply this within the world of work, there are, obviously, further considerations. For example, RSA can be used to create the key pair and for the signing and verification processes. However, with the specific padding standard PCKS 1.5, it is inadvisable to do so. The reason for this being that there is a specific attack which can be completed on the basis of this padding. In saying that, it's very common to come across RSA with this particular type of padding being used. So if there's a compatibility issue, then you need to look at the risks there and determine whether that's an appropriate thing to do. Of course, that's not to say that RSA is never usable.

There are other standards which can be used, which vary how the padding and other elements are completed in order to allow RSA to be used effectively. For

example, there is RSA-PSS with the updated PKCS number 2.1. This approach, instead, encodes using PSS and adds an element of randomness as well. So there's more complexity before the actual signing takes place. But the signing itself is as expected. It's just in the preparation of the data before it's digitally signed that changes.

Currently, the best approach is the Edwards-curve digital signature algorithm, EdDSA. This is out of scope within this module. Obviously, we can go into incredible depth in terms of cryptography and not, least of all, digital signatures. So if you are interested in more of the detail behind these particular aspects, I'll leave a reference below the video that you can follow-up if you're so inclined. That's all for this video. I hope you enjoyed it. And I'll see you next time.