SE 4472

Information Security

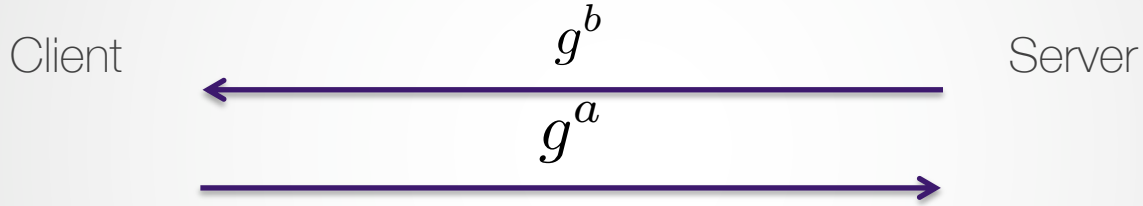
Proving Identity with Digital Certificates

Prof. Aleksander Essex

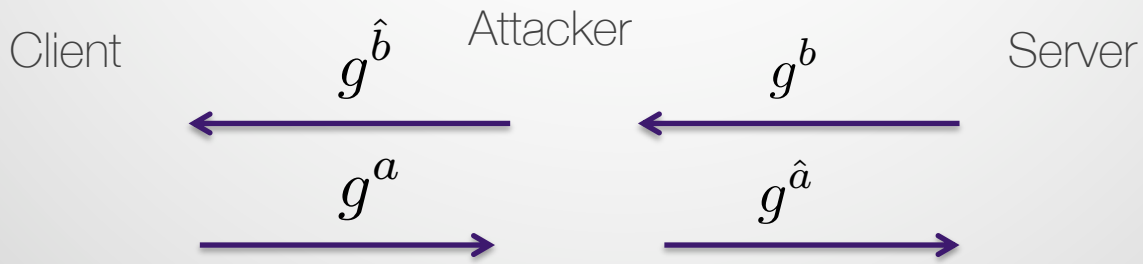
Western Engineering

How Signatures are used in TLS

- Suppose two parties want to establish a shared secret, with, say Diffie-Hellman

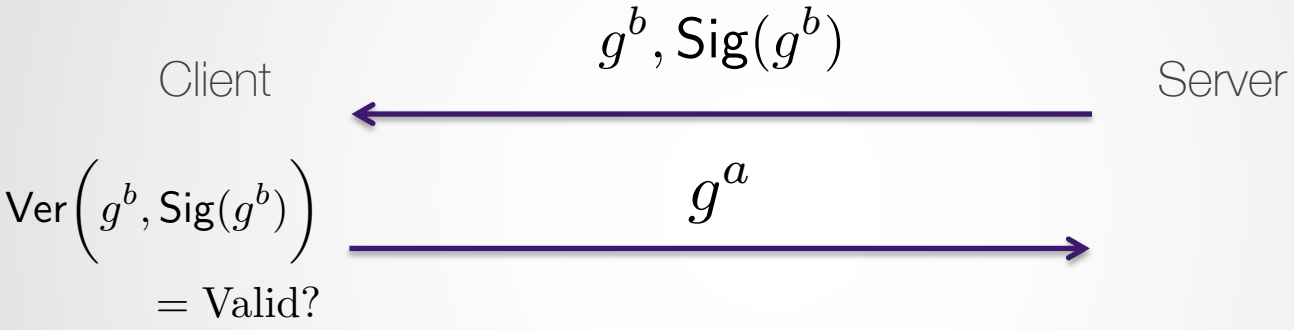


- How do the parties prevent man-in-the-middle attacks designed to swap out their public keys for the attacker's?



How Signatures are used in TLS

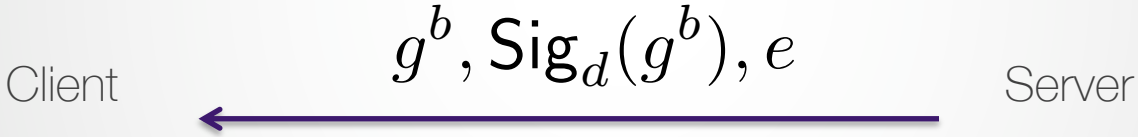
- Idea: The server can sign its public key



- Now the attacker must forge signature

Getting the Verification Key

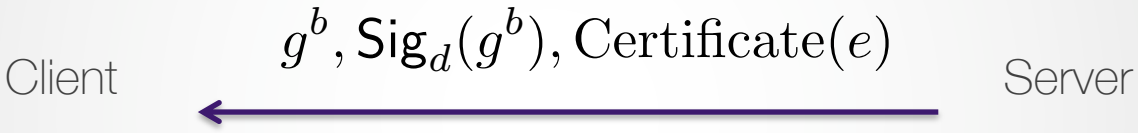
- Ok this seems like it could work. But first the client needs to get the server's verification key *e*. So how do they get it? The server could send it in-band:



- But then the attacker could swap out the server's *e* for their own *ê*:

Certificates

- What if there was a way to certify that the server's public key really was *e*?

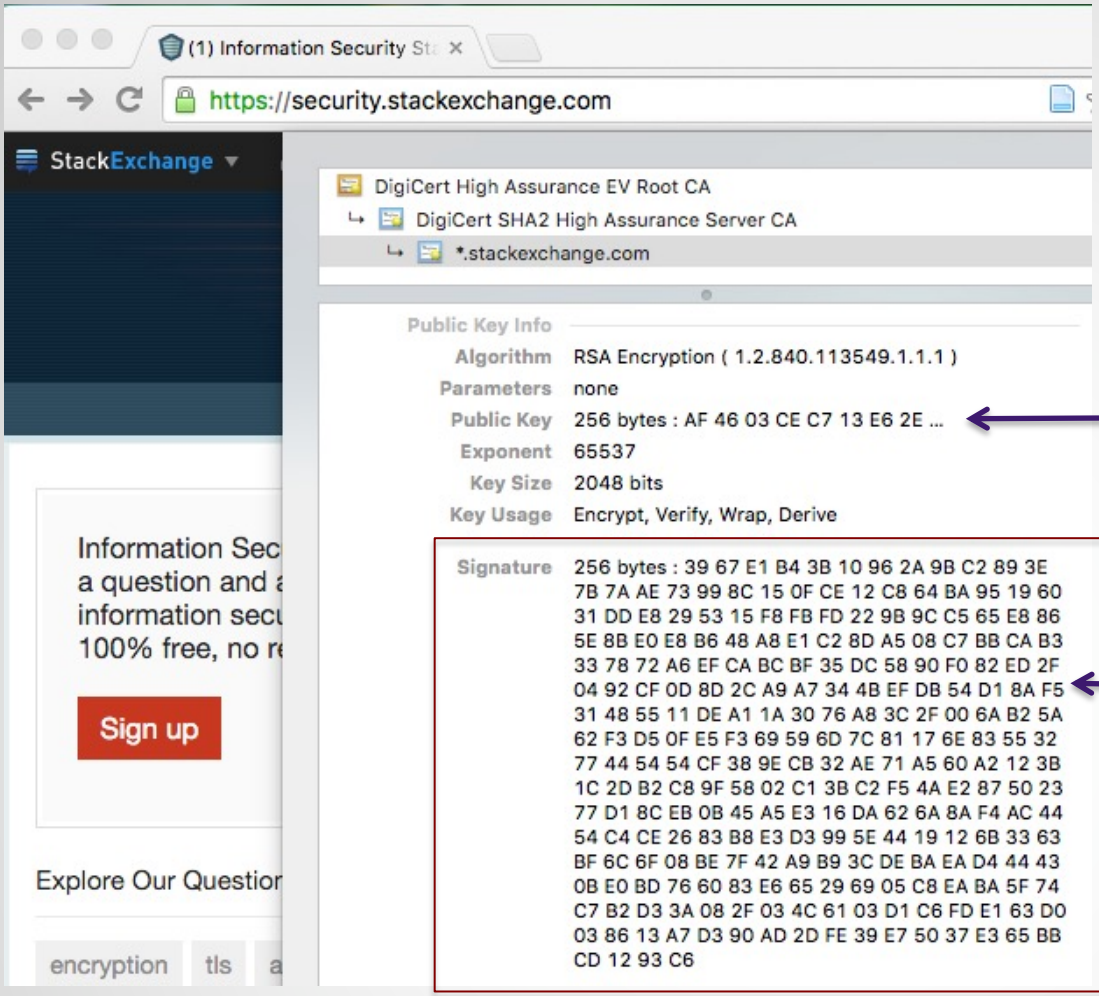


- The certificate could say something like "We the undersigned to solemnly pledge that the server's key is *e*."
- This "certificate authority" can even cryptographically protect this certificate by signing it

An Worked Example of Certificates and Signatures

Certificates

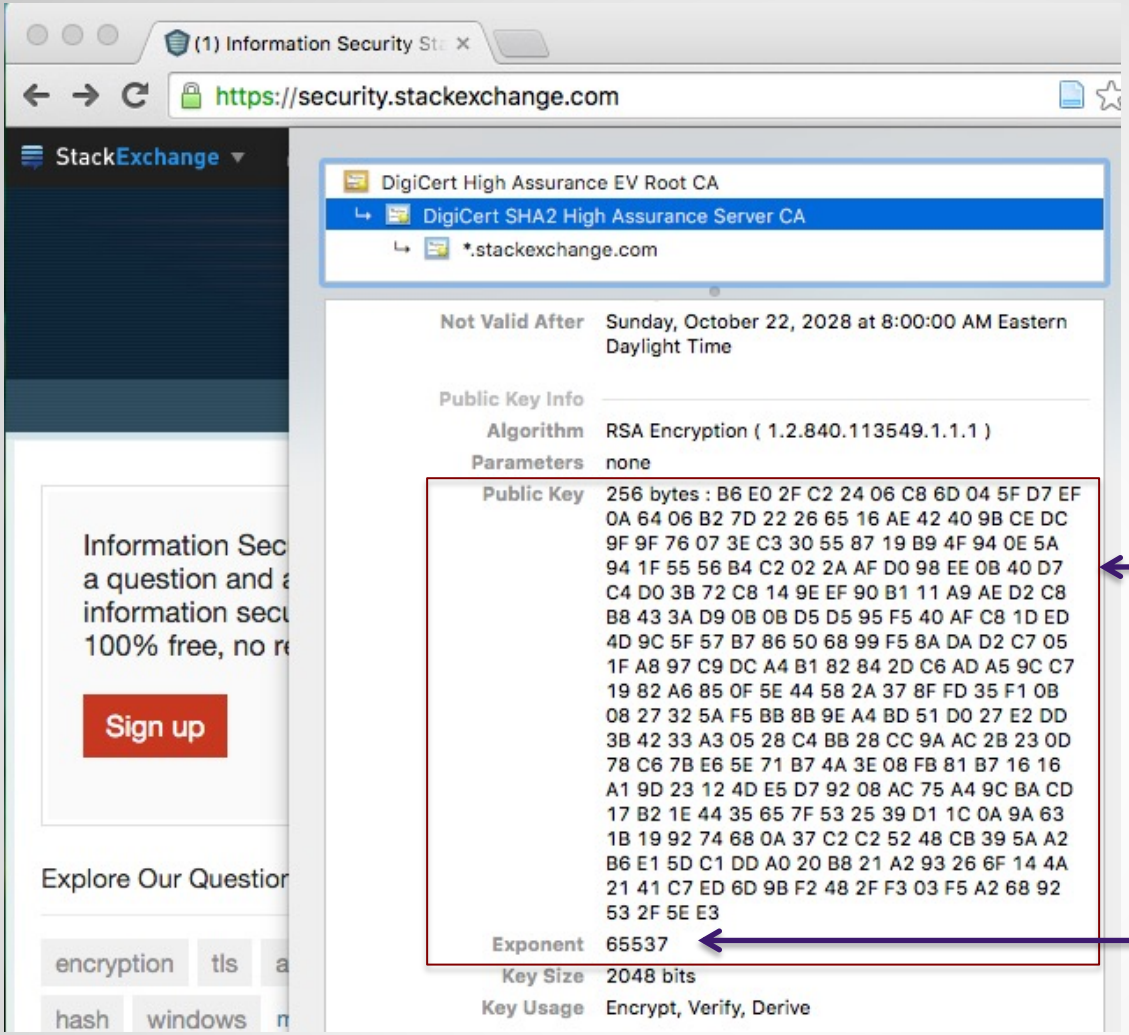
- Let's introduce certificates and solidify our understanding of certificates by doing a worked example
- We're going to take the certificate of stackexchange.com and verify the certificate authority's signature
- You can try this yourself with Python and OpenSSL



stackexchange.com public key

Signature on stackexchange's certiciate by DigiCert SHA2 High Assurance CA

sig=
39 67 E1 B4 3B 10 96 2A 9B C2 89 3E 7B 7A AE 73 99 8C 15 0F CE 12 C8 64 BA 95 19 60 31 DD E8 29 53 15 F8 FB FD 22 9B 9C C5 65 E8 86 5E 8B E0 E8 B6 48 A8 E1 C2 8D A5 08 C7 BB CA B3 33 78 72 A6 EF CA BC BF 35 DC 58 90 F0 82 ED 2F 04 92 CF 0D 8D 2C A9 A7 34 4B EF DB 54 D1 8A F5 31 48 55 11 DE A1 1A 30 76 A8 3C 2F 00 6A B2 5A 62 F3 D5 0F E5 F3 69 59 6D 7C 81 17 6E 83 55 32 77 44 54 54 CF 38 9E CB 32 AE 71 A5 60 A2 12 3B 1C 2D B2 C8 9F 58 02 C1 3B C2 F5 4A E2 87 50 23 77 D1 8C EB 0B 45 A5 E3 16 DA 62 6A 8A F4 AC 44 54 C4 CE 26 83 B8 E3 D3 99 5E 44 19 12 6B 33 63 BF 6C 6F 08 BE 7F 42 A9 B9 3C DE BA EA D4 44 43 0B E0 BD 76 60 83 E6 65 29 69 05 C8 EA BA 5F 74 C7 B2 D3 3A 08 2F 03 4C 61 03 D1 C6 FD E1 63 D0 03 86 13 A7 D3 90 AD 2D FE 39 E7 50 37 E3 65 BB CD 12 93 C6



DigiCert's modulus *n*

DigiCert's public exponent *e*

n=
B6 E0 2F C2 24 06 C8 6D 04 5F D7 EF 0A 64 06 B2 7D 22 26 65 16 AE 42 40 9B CE DC 9F 9F 76 07 3E C3 30 55 87 19 B9 4F 94 0E 5A 94 1F 55 56 B4 C2 02 2A AF D0 98 EE 0B 40 D7 C4 D0 3B 72 C8 14 9E EF 90 B1 11 A9 AE D2 C8 B8 43 3A D9 0B 0B D5 D5 95 F5 40 AF C8 1D ED 4D 9C 5F 57 B7 86 50 68 99 F5 8A DA D2 C7 05 1F A8 97 C9 DC A4 B1 82 84 2D C6 AD A5 9C C7 19 82 A6 85 0F 5E 44 58 2A 37 8F FD 35 F1 0B 08 27 32 5A F5 BB 8B 9E A4 BD 51 D0 27 E2 DD 3B 42 33 A3 05 28 C4 BB 28 CC 9A AC 2B 23 0D 78 C6 7B E6 5E 71 B7 4A 3E 08 FB 81 B7 16 16 A1 9D 23 12 4D E5 D7 92 08 AC 75 A4 9C BA CD 17 B2 1E 44 35 65 7F 53 25 39 D1 1C 0A 9A 63 1B 19 92 74 68 0A 37 C2 C2 52 48 CB 39 5A A2 B6 E1 5D C1 DD A0 20 B8 21 A2 93 26 6F 14 4A 21 41 C7 ED 6D 9B F2 48 2F F3 03 F5 A2 68 92 53 2F 5E E3

Result

Using the values from the previous slide, we compute:

$$sig^e \mod n$$

and get:

0001ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ff003031
300d0609608648016503040201050004203d47a154fc239c87a5a47cb03e93d2defe7c4728a129f27007e2bd08ecaefcce

Breakdown:

- Red: PKCS1 1.5 Padding
- Blue: ASN.1 Encoding header specifying the hash algorithm (SHA 256 in this case)
- Green: The (256-bit) hash of certificate

Result

- Download the certificate and save to cert.pem:

```
echo "" | openssl s_client -host stackexchange.com -port 443 | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p'  > cert.pem
```

- Grab the part of the certificate that gets signed:

```
$ openssl asn1parse -in cert.pem -out tobehashed -noout -strparse 4
```

- SHA256 hash it:

```
$ openssl sha256 tobehashed
SHA256(tbs)= 3d47a154fc239c87a5a47cb03e93d2defe7c4728a129f27007e2bd08ecaefcce
```

- Compare certificate hash (above) to hash recovered from signature (previous slide). They match!