Public Key Cryptography Basics

- **Authentication**: You want to be sure you communicate with the right person, not someone else pretending.
- **Authenticity**: You can verify that the information comes from the claimed source.
- **Integrity**: You must ensure that no one changes the data you exchange.
- **Confidentiality**: You want to prevent an unauthorised party from eavesdropping on your conversations.

Common Use of Asymmetric Encryption

Asymmetric encryption is relatively slow compared to symmetric encryption; therefore, we rely on asymmetric encryption to negotiate and agree on symmetric encryption ciphers and keys.

RSA

RSA is a public-key encryption algorithm that enables secure data transmission over insecure channels. With an insecure channel, we expect adversaries to eavesdrop on it.

The Math That Makes RSA Secure

RSA is based on the mathematically difficult problem of factoring a large number. Multiplying two large prime numbers is a straightforward operation; however, finding the factors of a huge number takes much more computing power.

Numerical Example

the modulo operation plays a significant role in cryptography. In the following simplified numerical example, we see the RSA algorithm in action:

1. Bob chooses two prime numbers: $p = 157$ and $q = 199$. He calculates $n = p \times q = 31243$.
2. With $\phi(n) = n - p - q + 1 = 31243 - 157 - 199 + 1 = 30888$, Bob selects $e = 163$ such that e is relatively prime to $\phi(n)$; moreover, he selects $d = 379$, where $e \times d = 1 \bmod \phi(n)$, i.e., $e \times d = 163 \times 379 = 61777$ and 61777 mod 30888 = 1. The public key is (n,e), i.e., (31243,163) and the private key is $(n,d), i.e., (31243,379).
3. Let's say that the value they want to encrypt is $x = 13$, then Alice would calculate and send *y* = *x*__*e* mod *n* = 13163 mod 31243 = 16341.
4. Bob will decrypt the received value by calculating *x* = *y*__*d* mod *n* = 16341379 mod 31243 = 13. This way, Bob recovers the value that Alice sent.

RSA in CTFs

tools for defeating RSA challenges in CTFs. My favourite is RsaCtfTool, which has worked well for me. rsatool.

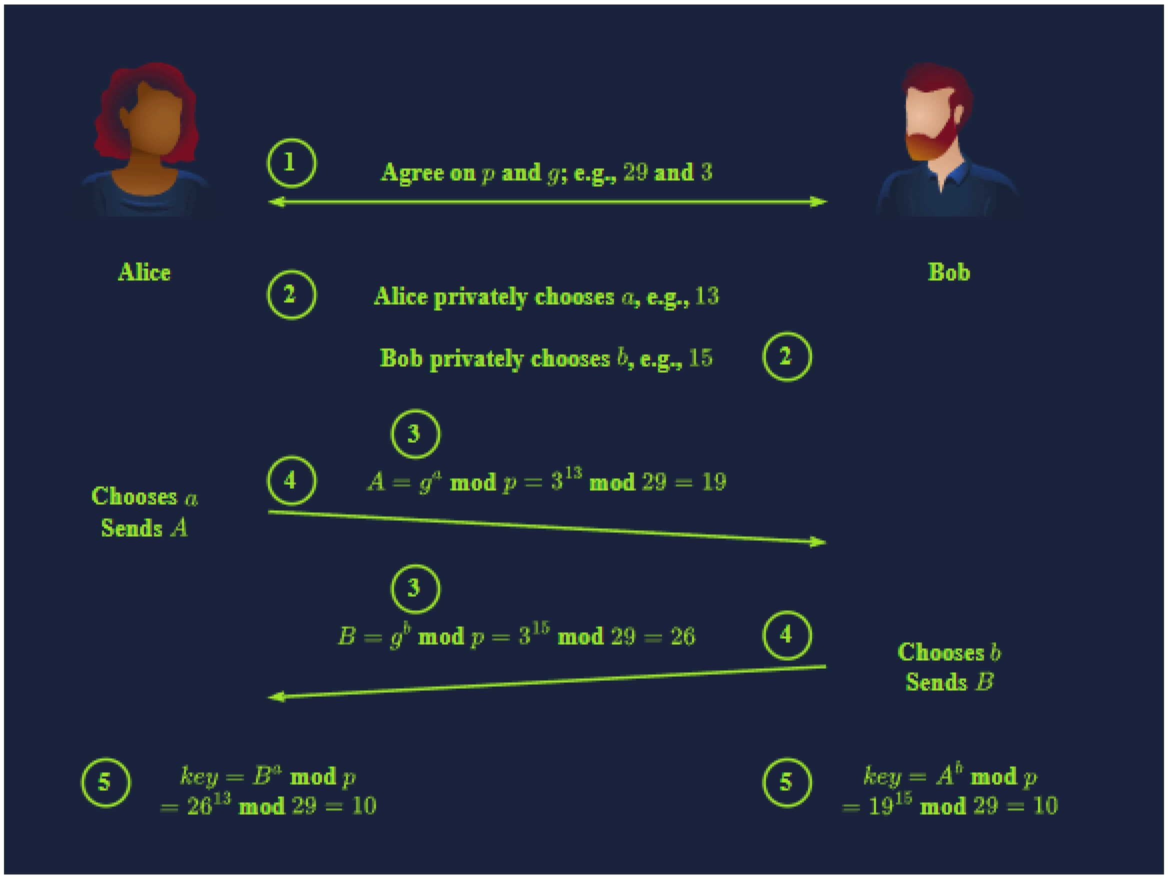
You need to know the main variables for RSA in CTFs: p, q, m, n, e, d, and c. As per our numerical example:

- p and q are large prime numbers
- n is the product of p and q
- The public key is n and e
- The private key is n and d
- m is used to represent the original message, i.e., plaintext
- c represents the encrypted text, i.e., ciphertext

Diffie-Hellman Key Exchange

Key exchange aims to establish a shared secret between two parties. It is a method that allows two parties to establish a shared secret over an insecure communication channel without requiring a pre-existing shared secret and without an observer being able to get this key

1. Alice and Bob agree on the **public variables**: a large prime number *p* and a generator *g*, where 0 < *g* < *p*. These values will be disclosed publicly over the communication channel. Although insecurely small, we will choose *p* = 29 and *g* = 3 to simplify our calculations.
2. Each party chooses a private integer. As a numerical example, Alice chooses *a* = 13, and Bob chooses *b* = 15. Each of these values represents a **private key** and must not be disclosed.
3. It is time for each party to calculate their **public key** using their private key from step 2 and the agreed-upon public variables from step 1. Alice calculates *A* = *g*__*a* mod *p* = 313 mod 29 = 19 and Bob calculates *B* = *g*__*b* mod *p* = 315 mod 29 = 26. These are the public keys.
4. Alice and Bob send the keys to each other. Bob receives *A* = *g*__*a* mod *p* = 19, i.e., Alice's public key. And Alice receives *B* = *g*__*b* mod *p* = 26, i.e., Bob's public key. This step is called the **key exchange**.
5. Alice and Bob can finally calculate the **shared secret** using the received public key and their own private key. Alice calculates *B*__*a* mod *p* = 2613 mod 29 = 10 and Bob calculates *A*__*b* mod *p* = 1915 mod 29 = 10. Both calculations yield the same result, *gab* mod *p* = 10, the shared secret key.


Alice
Bob
1 Agree on p and g; e.g., 29 and 3
2 Alice privately chooses a, e.g., 13
Bob privately chooses b, e.g., 15 2
3
Chooses a Sends A
4 $A = g^a$ mod $p = 3^{13}$ mod $29 = 19$
3
$B = g^b$ mod $p = 3^{15}$ mod $29 = 26$ 4
Chooses b Sends B
5 $key = B^a$ mod p $= 26^{13}$ mod $29 = 10$
5 $key = A^b$ mod p $= 19^{15}$ mod $29 = 10$

Diffie-Hellman Key Exchange is often used alongside RSA public key cryptography. Diffie-Hellman is used for key agreement, while RSA is used for digital signatures, key transport, and authentication, among many others.

SSH

Authenticating the Server

SSH client confirms whether we recognise the server's public key fingerprint. ED25519 is the public-key algorithm used for digital signature generation and verification

```
root@TryHackMe# ssh 10.10.244.173
The authenticity of host '10.10.244.173 (10.10.244.173)' can't be established.
ED25519 key fingerprint is SHA256:lLzhZc7YzRBDchm02qTX0qsLqeeiTCJg5ipOT0E/YM8.
This key is not known by any other name.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.244.173' (ED25519) to the list of known hosts.
```

Authenticating the Client

 In many cases, SSH users are authenticated using usernames and passwords like you would log in to a physical machine. However, considering the inherent issues with passwords, this does not fall within the best security practices.

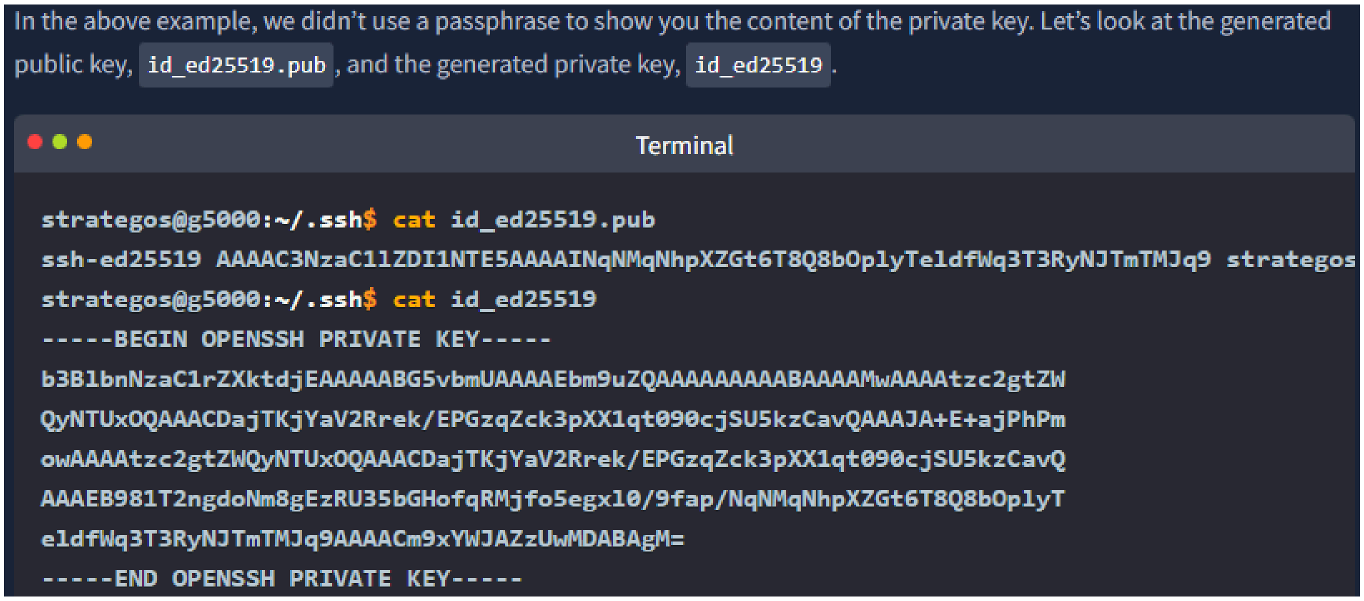
At some point, one will surely hit a machine with SSH configured with key authentication instead.

`ssh-keygen` is the program usually used to generate key pairs. It supports various algorithms, as shown on its manual page below.

```
root@TryHackMe# man ssh-keygen
[...]
-t dsa | ecdsa | ecdsa-sk | ed25519 | ed25519-sk | rsa
Specifies the type of key to create. The possible values are “dsa”, “ecdsa”, “ecdsa-sk”,
[...]
```

- **DSA (Digital Signature Algorithm)** is a public-key cryptography algorithm specifically designed for digital signatures.
- **ECDSA (Elliptic Curve Digital Signature Algorithm)** is a variant of DSA that uses elliptic curve cryptography to provide smaller key sizes for equivalent security.
- **ECDSA-SK (ECDSA with Security Key)** is an extension of ECDSA. It incorporates hardware-based security keys for enhanced private key protection.
- **Ed25519** is a public-key signature system using EdDSA (Edwards-curve Digital Signature Algorithm) with Curve25519.
- **Ed25519-SK (Ed25519 with Security Key)** is a variant of Ed25519. Similar to ECDSA-SK, it uses a hardware-based security key for improved private key protection.

```
root@TryHackMe# ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/strategos/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/strategos/.ssh/id_ed25519
Your public key has been saved in /home/strategos/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:4S4DQvRfp52UuNwg++nTcWlnITEJTbMcCU0N8UYC1do strategos@g5000
The key's random art image is:
+--[ED25519 256]--+
|  .       +XXB.  |
| . .     . oBBo  |
|  . . . = + o=o  |
| .   . * X .o.E  |
|  . . o S +  o . |
|   . . o .. + o  |
|      o +. + o   |
|       +. .      |
|        ..       |
+----[SHA256]-----+
```


In the above example, we didn't use a passphrase to show you the content of the private key. Let's look at the generated public key, `id_ed25519.pub`, and the generated private key, `id_ed25519`.

```
strategos@g5000:~/.ssh$ cat id_ed25519.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAINqNMqNhpXZGt6T8Q8bOplyTeldfWq3T3RyNJTmTMJq9 strategos
strategos@g5000:~/.ssh$ cat id_ed25519
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW
QyNTUxOQAAACDajTKjYaV2Rrek/EPGzqZck3pXX1qt090cjSU5kzCavQAAAJA+E+ajPhPm
owAAAAtzc2gtZWQyNTUxOQAAACDajTKjYaV2Rrek/EPGzqZck3pXX1qt090cjSU5kzCavQ
AAAEB981T2ngdoNm8gEzRU35bGHofqRMjfo5egxl0/9fap/NqNMqNhpXZGt6T8Q8bOplyT
eldfWq3T3RyNJTmTMJq9AAAACm9xYWJAZzUwMDABAgM=
-----END OPENSSH PRIVATE KEY-----
```

SSH Private Keys

Using tools like John the Ripper, you can attack an encrypted SSH key to attempt to find the passphrase, highlighting the importance of using a complex passphrase and keeping your private key private.

When generating an SSH key to log in to a remote machine, you should generate the keys on your machine and then copy the public key over, as this means the private key never exists on the target machine using `ssh-copy-id`

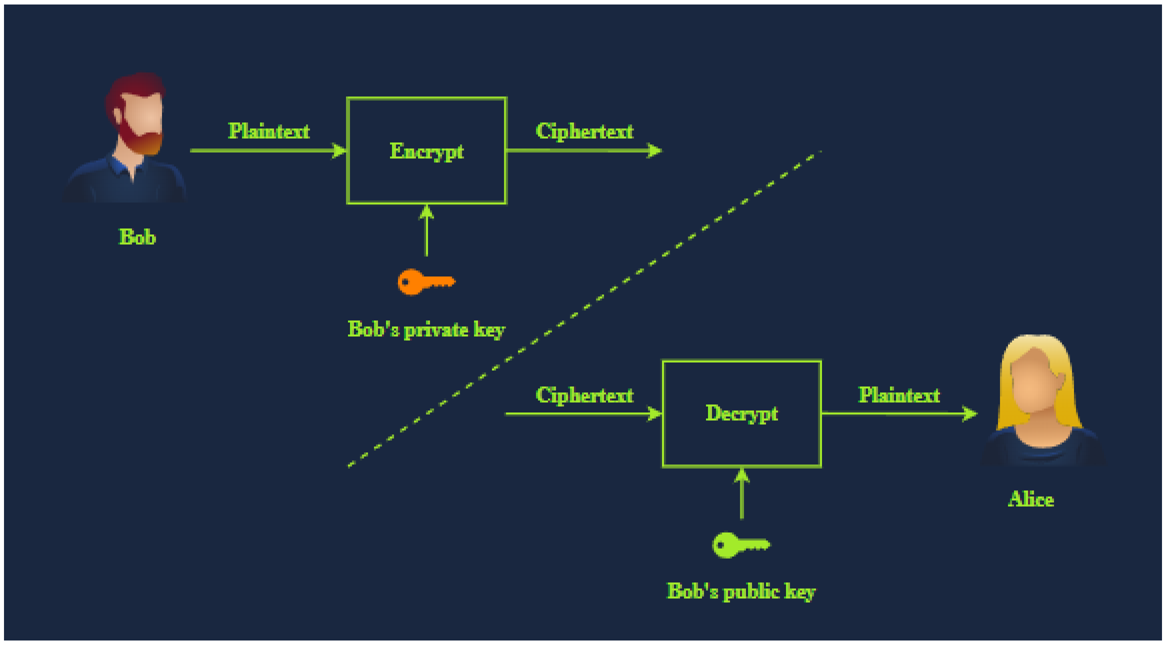
`ssh -i privateKeyFileName user@host` is how you specify a key for the standard Linux OpenSSH client.

Using SSH Keys to Get a "Better Shell"

During CTFs, penetration testing, and red teaming exercises, SSH keys are an excellent way to "upgrade" a reverse shell, assuming the user has login enabled. Note that www-data usually does not allow this, but regular users and root will work. Leaving an SSH key in the `authorized_keys` file on a machine can be a useful backdoor, and you don't need to deal with any of the issues of unstabilised reverse shells like Control-C or lack of tab completion.

Digital Signatures and Certificates

Digital signatures provide a way to verify the authenticity and integrity of a digital message or document. Using asymmetric cryptography, you produce a signature with your private key, which can be verified using your public key



Certificates: Prove Who You Are!

Certificates are an essential application of public key cryptography, and they are also linked to digital signatures. A common place where they're used is for HTTPS

The web server has a certificate that says it is the real tryhackme.com. The certificates have a chain of trust, starting with a root CA (Certificate Authority)

you can get your own TLS certificates for domains you own using Let's Encrypt for free

PGP and GPG

PGP stands for Pretty Good Privacy. It's software that implements encryption for encrypting files, performing digital signing, and more. GnuPG or GPG is an open-source implementation of the OpenPGP standard.

GPG is commonly used in email to protect the confidentiality of the email messages. Furthermore, it can be used to sign an email message and confirm its integrity.

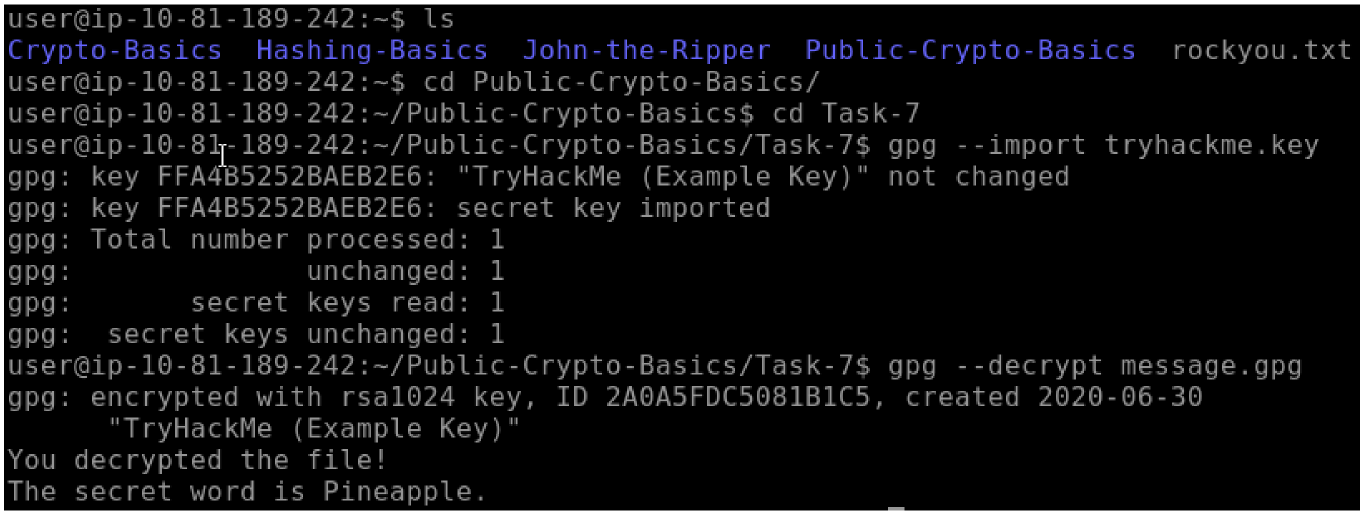
```
gpg --full-gen-key
gpg (GnuPG) 2.4.4; Copyright (C) 2024 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
   (1) RSA and RSA
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
   (9) ECC (sign and encrypt) *default*
  (10) ECC (sign only)
  (14) Existing key from card
Your selection? 9
Please select which elliptic curve you want:
   (1) Curve 25519 *default*
   (4) NIST P-384
   (6) Brainpool P-256
Your selection? 1
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: strategos
Email address: strategos@tryhackme.thm
[...]
pub   ed25519 2024-08-29 [SC]
      AB7E6AA87B6A8E0D159CA7FFE5E63DBD5F83D5ED
uid                      Strategos <strategos@tryhackme.thm>
sub   cv25519 2024-08-29 [E]
```

may need to use GPG to decrypt files in CTFs. With PGP/GPG, private keys can be protected with passphrases in a similar way that we protect SSH private keys. If the key is passphrase protected, you can attempt to crack it using John the Ripper and `gpg2john`



```
user@ip-10-81-189-242:~$ ls
Crypto-Basics  Hashing-Basics  John-the-Ripper  Public-Crypto-Basics  rockyou.txt
user@ip-10-81-189-242:~$ cd Public-Crypto-Basics/
user@ip-10-81-189-242:~/Public-Crypto-Basics$ cd Task-7
user@ip-10-81-189-242:~/Public-Crypto-Basics/Task-7$ gpg --import tryhackme.key
gpg: key FFA4B5252BAEB2E6: "TryHackMe (Example Key)" not changed
gpg: key FFA4B5252BAEB2E6: secret key imported
gpg: Total number processed: 1
gpg:              unchanged: 1
gpg:       secret keys read: 1
gpg:  secret keys unchanged: 1
user@ip-10-81-189-242:~/Public-Crypto-Basics/Task-7$ gpg --decrypt message.gpg
gpg: encrypted with rsa1024 key, ID 2A0A5FDC5081B1C5, created 2020-06-30
      "TryHackMe (Example Key)"
You decrypted the file!
The secret word is Pineapple.
```

here i imported a key and used it to decrypt a gpg file to reveal the secret word