NoteHack++

WHEN CONVENIENCE MEETS SECURITY

GILAD WEISS & NERIA TZIDKANI

What will be covered in this presentation

- DLL Injection
- API Hooking
- Multi-tiered Key Management (like what *Ransomware* uses)
- Key Derivation – PBKDF2
- Crypto Algorithms – AES CTR, GCM, RSA

What will **NOT** be covered in this presentation

- How to create Ransomware

DLL Injection – what does it involve?

- Taking a snapshot of the current running processes.
- Finding the process who’s name fits the name of the program we want (“*NotePad*++”).
  - This gives us its PID

DLL Injection – what does it involve?

- Allocating memory in the target process that will contain the path of the DLL to be injected
  - We need to use it as an argument in a function called LoadLibrary

DLL Injection – what does it involve?

- `LoadLibrary` is in `kernel32.dll`
  - And therefore in every process
- We'll find the address of `LoadLibrary` in the target process.
  - This function can allocate memory for, and load a DLL into the program

DLL Injection – what does it involve?

- Creating a new thread in the target process which starts at the `LoadLibrary` function’s address, and has the allocated DLL path address as an argument.
  - This will initiate the loading of the DLL.

DLL Injection – what does it involve?

- The main function in the DLL will be invoked by the operating system.
  - From there we can force the target process to do anything we want.

API Hooking – what does it involve?

- Finding the functions that are optimal hooking candidates
- Messing with them

API Hooking – what does it involve?

Mission: Adding a `jmp` command to the original code

- We don't want to mess with the offsets of the entire program
  - Not fun. It's better to overwrite something of the same size
- Instead, we take note of the relative addresses of the commands we want to replace, as well as their respective size in bytes.
  - We will use this information when we overwrite them

API Hooking – what does it involve?

- In the injected DLL, we find the base address of the EXE that called it – our target program
- The absolute address is calculated by adding the relative address to the base address.
- The return address is calculated by adding the number of bytes we have overwritten to the absolute address.

API Hooking – what does it involve?

- Now that we have hooked the desired API’s, we create functions for the hook spots to jump to.
- **Problem:** registers may be changed due to our actions.

API Hooking – what does it involve?

- Now that we have hooked the desired API’s, we create functions for the hook spots to jump to.
- **Solution:** use naked inline assembly, so the compiler won’t add potentially destructive “fixes”.
  - Any registers that are directly affected by our actions need to be protected with push and pop

Now what?

- Now we can inject & hook without any crash or malfunction.
- With the technical part done, we need to start the actual securing of the editor
- We'll start with the design

Design

- How do we manage the keys for all the files?
  - We could give each file its own password and trust that the user remembers them all
  - We could use an online authentication server
  - We could save the keys in a secure enclave
  - We can have the user call us for his password whenever he needs it

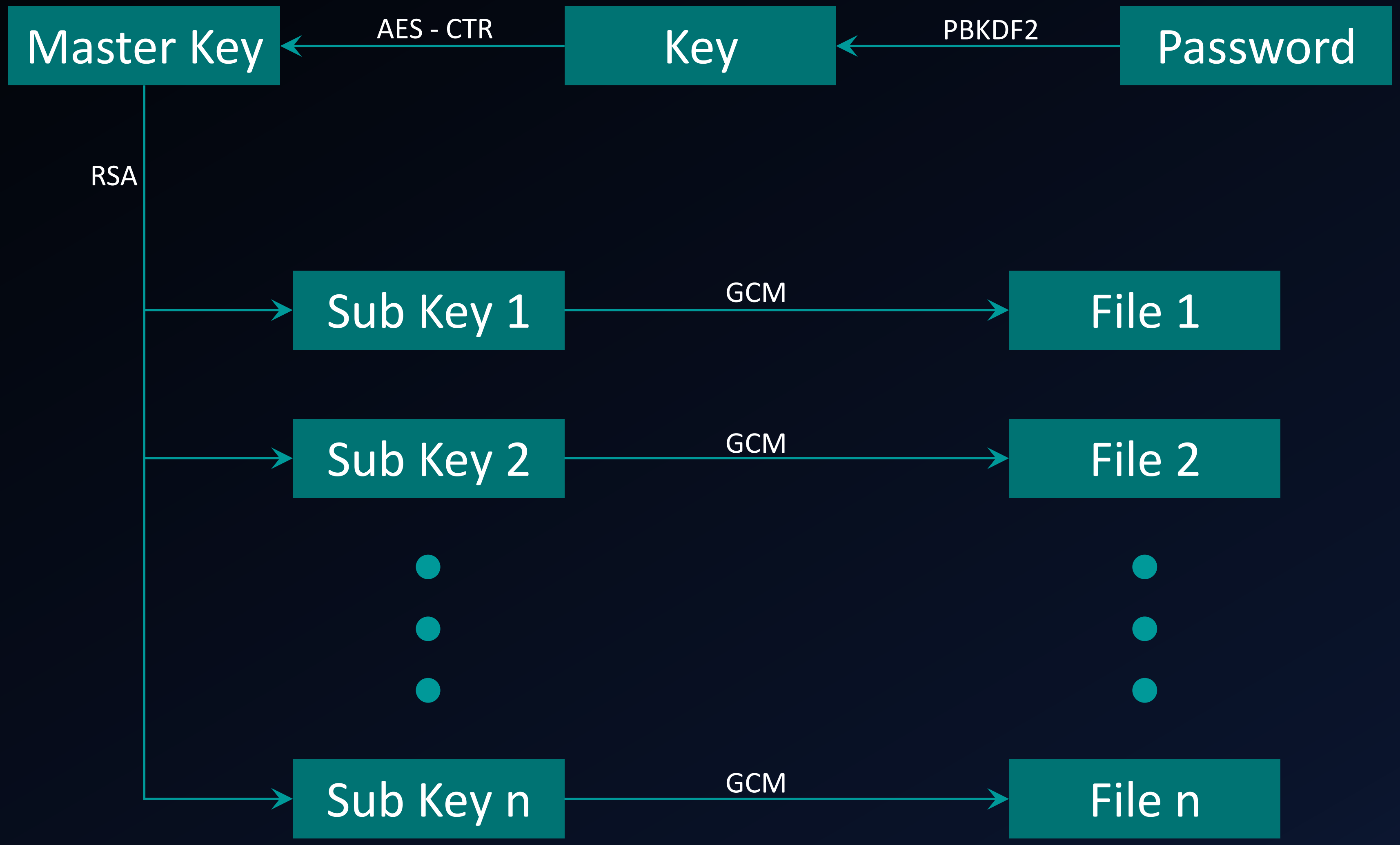
Design

- How do we manage the keys for all the files?
  - We could give each file its own password and trust that the user remembers them all
  - We could use an online authentication server
  - We could save the keys in a secure enclave
  - We can have the user call us for his password whenever he needs it
- All of the above options are ok (except maybe the last one)
- But isn’t there a better way?

Our multi-tiered key management system

- The user enters a password.
- The password *"magically"* becomes a key (discussed later)
  - for now, just remember the name PBKDF2
- This key protects the *"master key"* – a 2048-bit RSA key.
- The *"master key"* protects the "file keys" – randomly generated keys.
- Each key, encrypts a single file

Our multi-tiered key management system



Our multi-tiered key management system

Is that confused screaming I hear?

Let’s break it down

Key Derivation

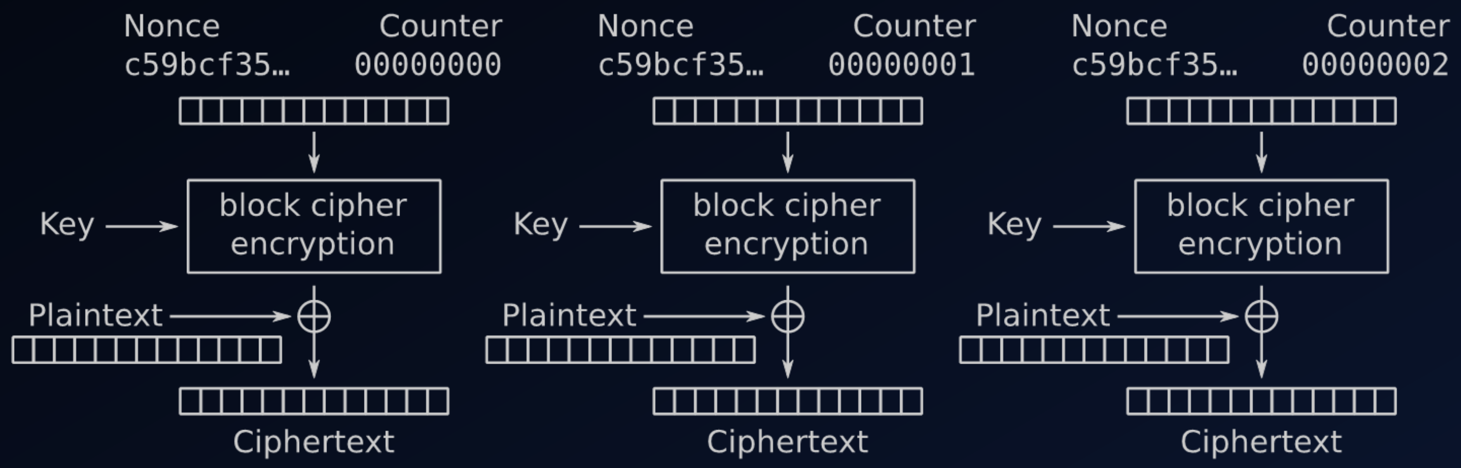
- Makes short passwords longer.
- Frustrate *Brute-Force* attacks
- As well as *Dictionary* attack

Key Derivation – PBKDF2

- Pros
  - Approved by NIST
  - Deliberately slow
  - Time tested
  - Is included in MbedTLS (our lib of choice)
- Cons
  - A good GPU will greatly reduce calculation time.
- `SHA-256` consumes more memory then `SHA-512`.
- A good iteration number for the average PC: 100,000.

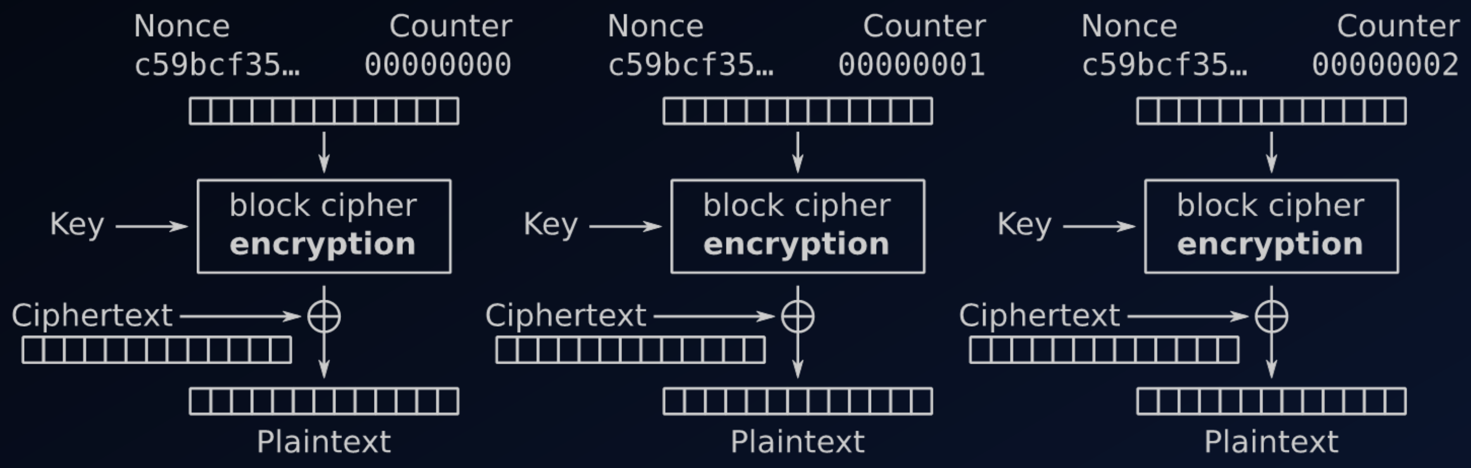
AES – Counter Mode

- Easy implementation.
- Faster than CBC Mode.
- Useful when there is no need for authentication



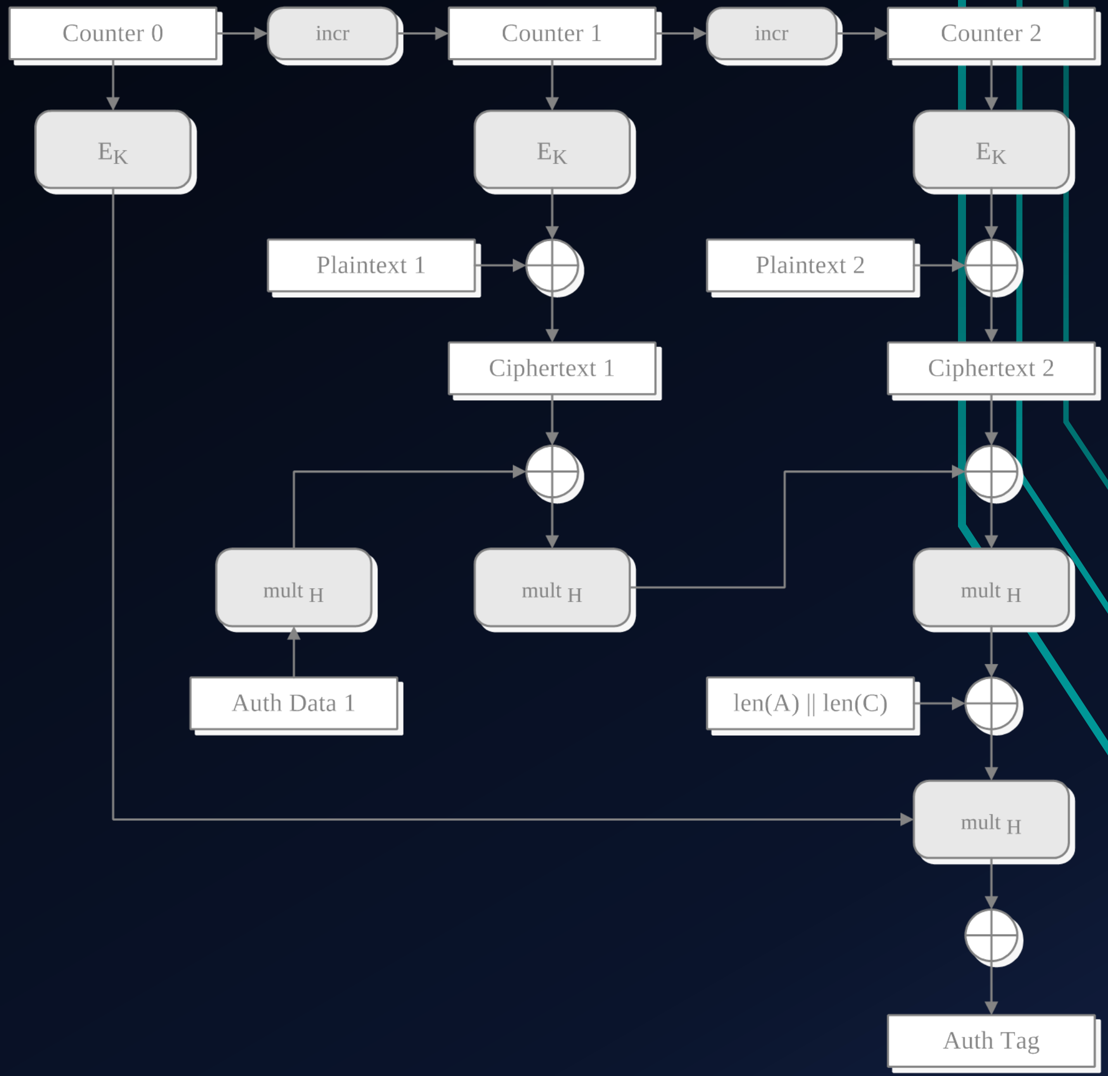
AES – Counter Mode

- Easy implementation.
- Faster than CBC Mode.
- Useful when there is no need for authentication



GCM – Galois Counter Mode

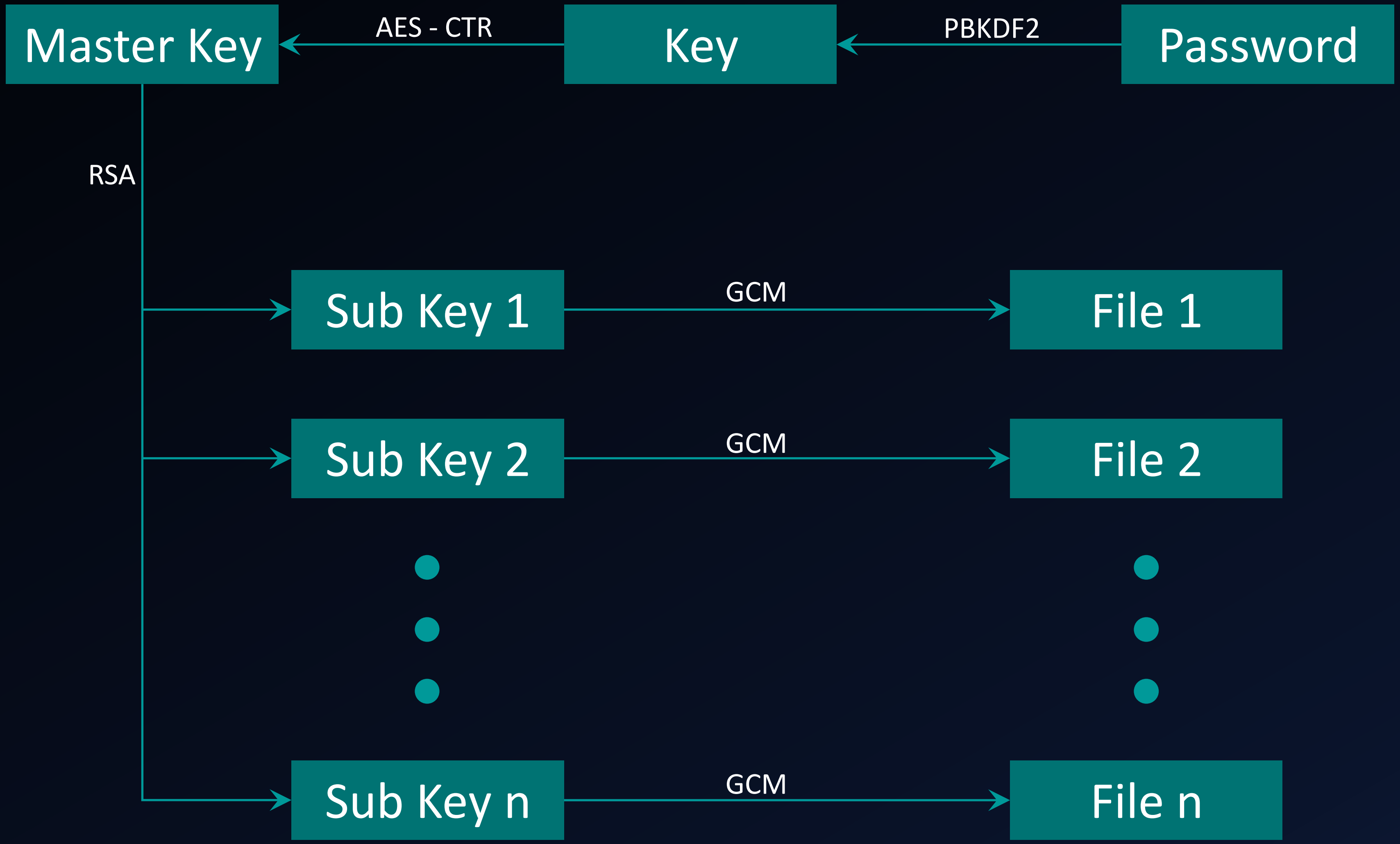
- High efficiency & performance
- Fast authentication
- If the tag is invalid, nothing will be decrypted



Breaking it down

- PBKDF2
  - Slow
  - Helps against *Brute-force* and *Dictionary* attacks
- AES-CTR
  - Fast and secure
  - No authentication
- GCM
  - Fast authentication

Our multi-tiered key management system



Our multi-tiered key management system

* screaming stops *

Isn't that better?

Our multi-tiered key management system

Why use three keys instead of one?

- Easy password changing
- Secure collaboration

Vulnerabilities

That sounds perfect!

A system nobody can hack?

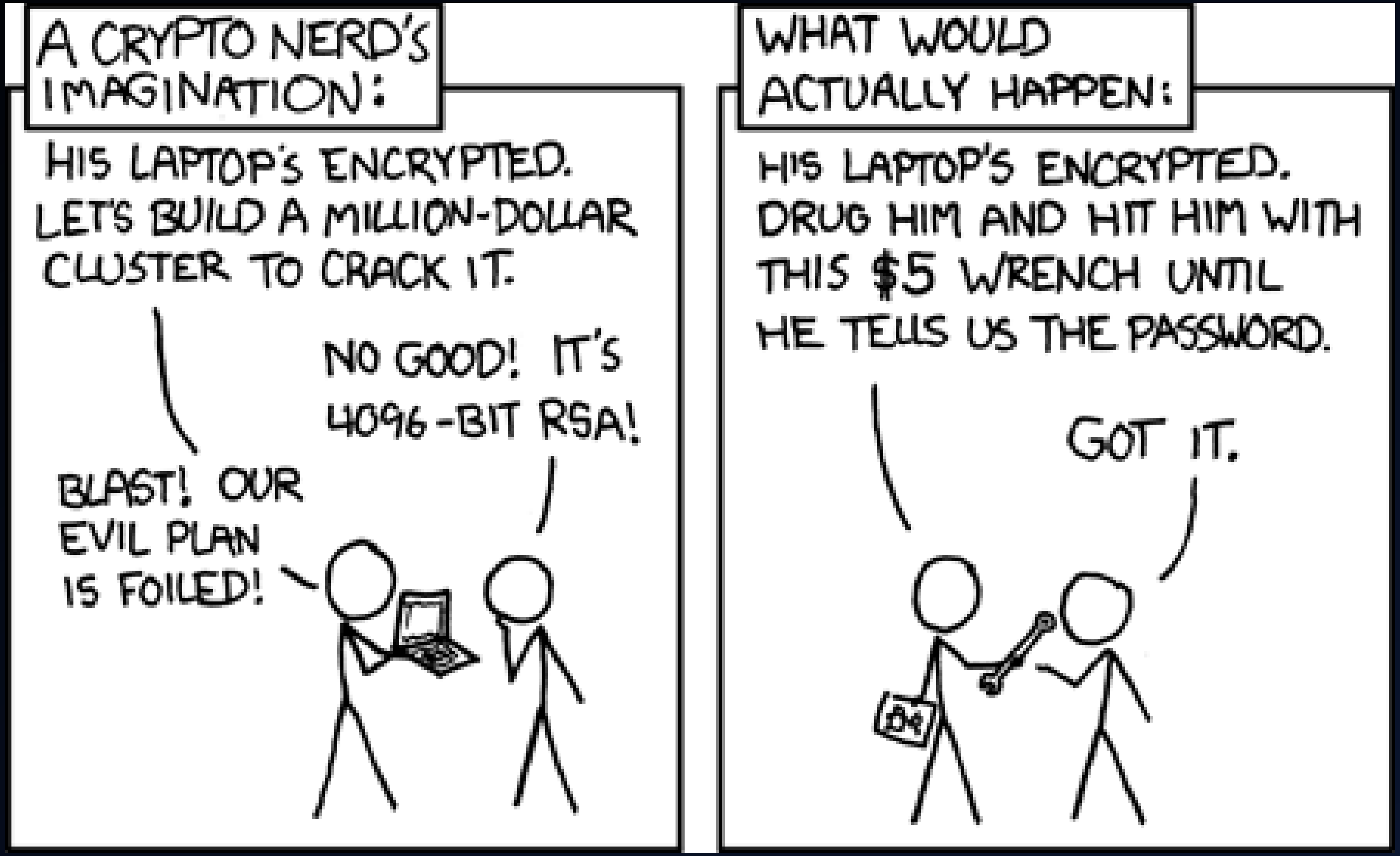
Well…

Vulnerabilities

- Memory dump.
- Another injection.
- A person behind your back.

- And…

Vulnerabilities



Future development

- Collaboration
  - Our multi-tiered key management system uses an asymmetric key to encrypt the file keys
  - Using a colleague’s public key, we can securely allow him to collaborate without revealing our key or password

Questions?