**Generating and encrypting**

I’m working through how a ransomware script generates a random AES encryption key for file encryption. This AES key is then encrypted with the attacker's public key, ensuring only the attacker's decryption key can unlock it.

Here’s the simplified logic in plain language:

1. **The Script Generates a Random AES Key**
   * AES is a fast, symmetric algorithm, so it’s used to quickly encrypt all the victim’s files.
   * Let’s call this key the **“AES key.”**
2. **The Script Encrypts Files With the AES Key**
   * Each file in Documents/Pictures is scrambled using AES (CBC mode).
   * Once done, it deletes the original file, leaving only the encrypted version.
3. **The Script Encrypts the AES Key Itself Using the RSA Public Key**
   * The RSA **public key** you pasted in the script is used for this step.
   * This means the AES key (which is the “real” key to unlock the files) gets locked inside RSA.
   * The encrypted AES key is written out as encrypted\_key.bin.
4. **Result**: Only Someone With the RSA Private Key Can Unlock the AES Key
   * Because the AES key is itself encrypted with the RSA public key, the **matching RSA private key** is required to decrypt (unlock) it.
   * Without the private key, the victim cannot recover the AES key, and therefore cannot decrypt any files.

**Why This Matters**

* If you only encrypted the files with RSA directly, it would be very slow for large files. Instead, you do a “hybrid” approach:
  1. Generate a random AES key for fast file encryption.
  2. Lock the AES key using RSA public key, so only the RSA private key can recover it.
* **Public Key**: Placed in the script, used for encryption.
* **Private Key**: Held by the attacker. Without it, the AES key remains locked in encrypted\_key.bin.
* **End**: Victim’s files are unreadable unless the private key is provided.