KEY RECOVERY OF PASSWORD-BASED AES-GCM WITH THE PARTITIONING ORACLE ATTACK

Members:
Christian James Tan (NUS High School of Mathematics and Science)
Xu Jingxin (Anglo-Chinese School (Independent))

Mentors:
Ruth Ng Ii Yung, Choo Jia Guang
(DSO National Laboratories)

1) BACKGROUND ON AUTHENTICATED ENCRYPTION & COLLISIONS

Authenticated Encryption (Symmetric scheme)

Encryption: ensures confidentiality (secrecy) - jumbles PT bytes to CT akin to random permutation

MAC: ensures authenticity& integrity – that the message has not been illicitly changed

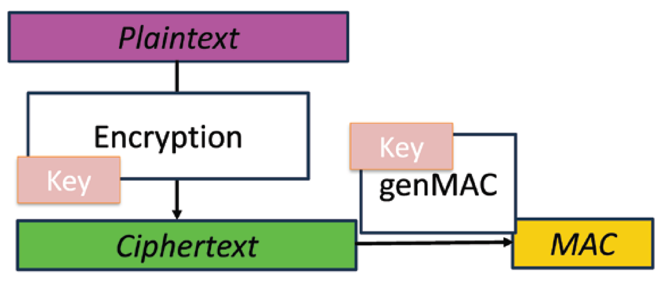

Fig 1.1: Encryption

Encryption (encrypt-then-MAC)

- Encrypt PT to CT
- Generate a MAC of the CT and send it with CT

Decryption

- Authentication: recalculate MAC– compare it with received MAC
- Decryption fails if they don't match
- Then decrypt CT to get PT

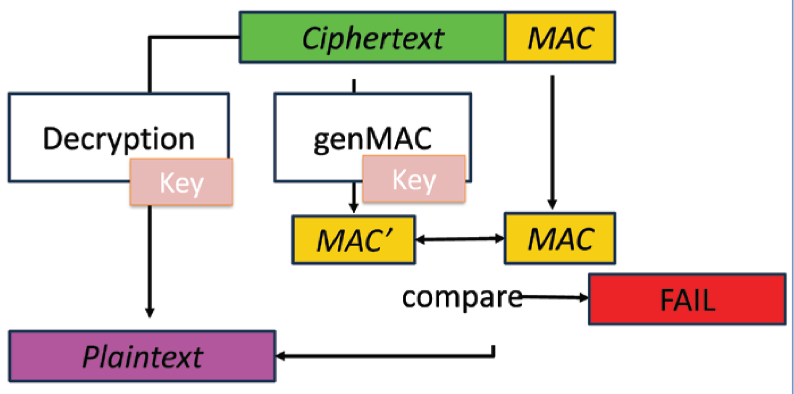

Fig 1.2: Decryption

Key multi-collision

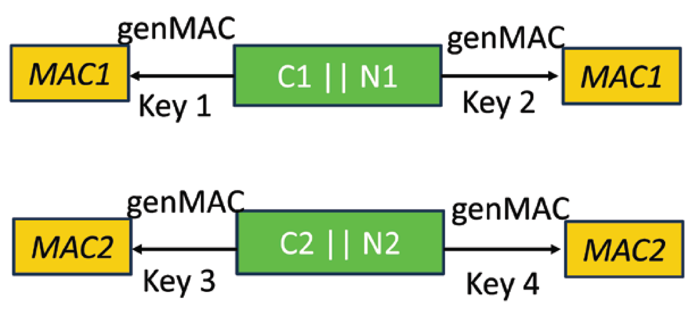

Fig 2: Collisions

- Same CT (and nonce) decrypts successfully under different keys.
- *Colliding* ciphertexts in *non-key-committing* schemes
- Multi-collisions: collide many keys

Examples of non-committing schemes

- GCM (Galois/Counter Mode)
- ChaCha20/Poly1305
- => MACs are polynomial based

2) PARTITIONING ORACLE ATTACK[1]

Attack target: the Partitioning Oracle

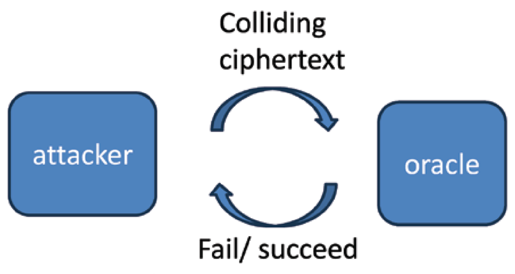

Fig 3: Partitioning Oracle

- Decryption oracle
- Indicates failure OR success of decryption.
- Some indicators eg error message, lack of response, timing side channels

Attack procedure

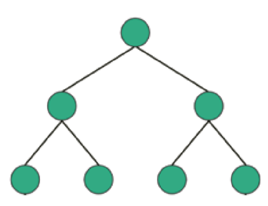

Fig 4: Binary tree

To leak the key used by a partitioning oracle:

- Compute a colliding ciphertext for a chosen set of 'colliding' keys.
- Query oracle and observe decryption status
  - If succeed: oracle's key is in set of 'colliding' keys
  - If fail: oracle's key is not in set (in complement of set).
- Compute next colliding ciphertext and repeat.
- => Attacker 'partitions' and narrows down the key space.
- Binary search (ideal) or linear scan in chunks followed by binary search (non-ideal)

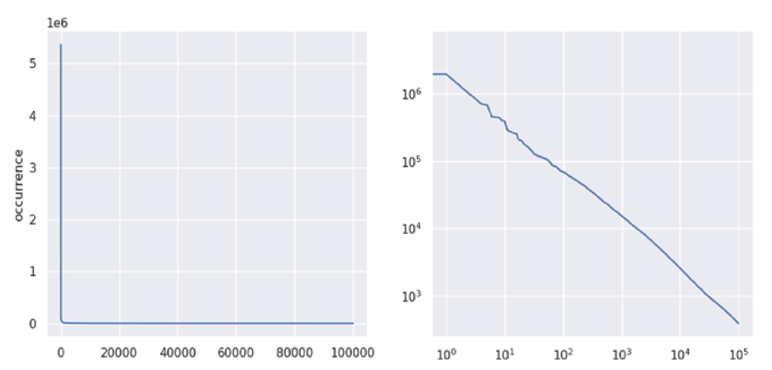

Fig 5: Zipfian distribution – rank and frequency in inverse proportion

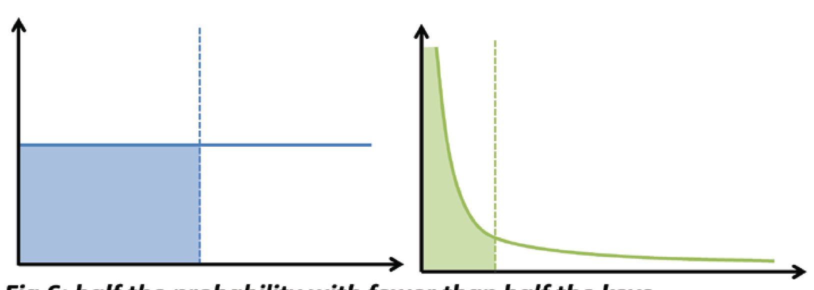

Fig 6: half the probability with fewer than half the keys.

Password-derived keys

- Some keys are derived from passwords using key derivation functions
- Passwords are non-uniformly distributed - instead, they follow a Zipfian distribution
- To partition the keyset in half (by probability), the colliding ciphertext requires fewer than half the keys
  - Faster & more feasible construction of ciphertexts

3) IMPLEMENTATION DETAILS OF ATTACK

Terms
PT: plaintext (original message)
CT: Ciphertext (undecipherable gibberish)
MAC: message authentication code
genMAC: function that generates it

Scenario description

- Messaging app
- Encrypt & sends messages using AES-GCM.
- Pre-shared passphrase used as the key in GCM
- Processing of ciphertext and MAC (and nonce): Authentication → decryption → decoding
- The receiver sends an acknowledgement message before exiting program (to confirm reception)
- => We can exploit this as our decryption oracle!

We used a password list of size 100 000 passwords with associated frequency data, and it takes an average of 11 guesses and a maximum of 17 to recover the key used, with collisions of up to 20,000 keys (<43ms).

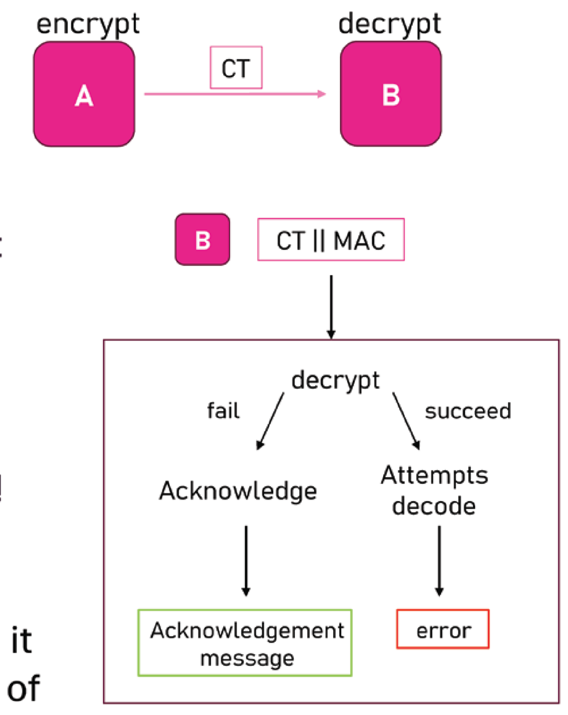

Fig 7: differentiating decryption failure/ success

Time complexity of attack

- For polynomial-based MACs e.g. GCM, construction of colliding ciphertexts is equivalent to polynomial interpolation:

Naïve

$$O(n^2) \Rightarrow \sum_{i=1}^{\log(n)} \frac{O(n^2)}{2^i} = O(n^2)$$

Fast

$$O(n \log^2 n) \Rightarrow \sum_{i=1}^{\log(n)} \frac{O(n \log^2 n)}{2^i} \approx O(n \log^2 n)$$

- In fastest case, still $O(n \log^2 n)$

- Dictionary attack: just try all passwords from most to least probable?
  - Runs in $O(n) < O(n \log^2 n) \Rightarrow$ better than partitioning?
- No. of queries:
  - Dictionary: $O(n)$
  - Partitioning Oracle Attack: $O(\log n)$
- Disregarding ciphertext construction, partitioning wins!
  - Precompute ciphertexts ⇒ Maximum speedup

How is this better than dictionary attack?

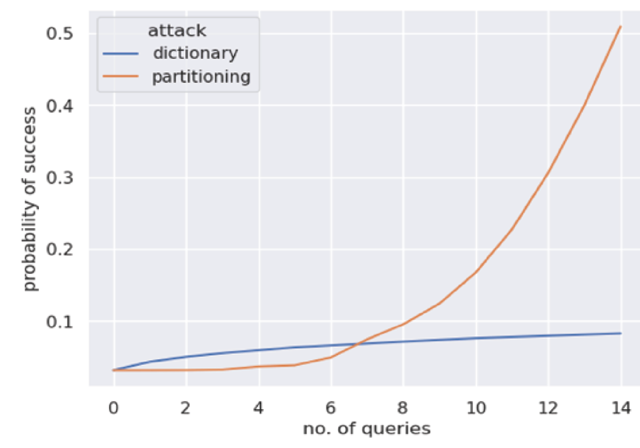

Fig 8: Partitioning oracle attack requires fewer queries

4) CONDITIONS, APPLICATIONS & MITIGATIONS

Condition 1 scheme used is non-key-committing

- Use committing schemes.Eg HMAC
- Make schemes committing CTX[2]

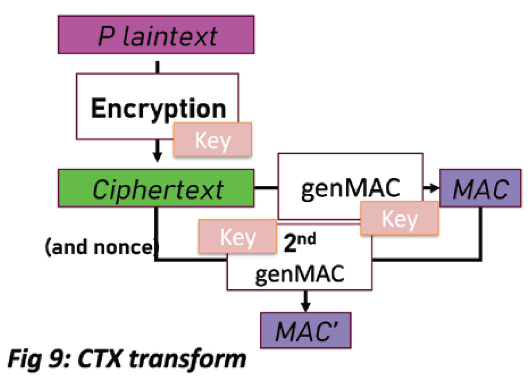

Fig 9: CTX transform

Condition 2 Access to partitioning oracle:

- Hide information on decryption failure/ success

Condition 3 (optional) Password-derived key

- Don't use a pre-shared passphrase
- Use uniformly generated random keys

Conclusion & further work: This attack **is a viable method of key**-recovery due to the speedup it provides when the 3 conditions are met, as shown by our example, and can be applied to TLS (pre-shared key and GCM mode) in future work.

Acknowledgements

We sincerely thank our mentors Ruth and Jia Guang, without whom this would have been impossible.

References

[1] Len, J., Grubbs, P., Ristenpart, T., 2021, Partitioning Oracle Attacks, 30th USENIX security symposium (USENIX Security 21): 195-212.

[2] Chan, J., Rogaway, P., 2022, On Committing Authenticated-Encryption, European Symposium on Research in Computer Security: 275-294.