Flexible Password-Based Encryption: Securing Cloud Storage and Provably Resisting Partitioning-Oracle Attacks

Ujjwal Prakash [2022202009]
Prashant Kumar [2022201058]
Rishabh Gupta [2022202011]

International Institute of Information Technology, Hyderabad, India

21st April 2023

Introduction

- FPBE is a method to encrypt data using a password and provides authenticity which is used for end-to-end security that protects against a malicious server.
- Used in modern applications such as end-to-end secure cloud storage.
- Allows reusing of the nonce, associated data and salts, making it easier to use in practice.
- Ensures that the data is not tampered by an unauthorized entity.

Traditional Password-Based Encryption (TPBE)

- Encrypts a message M using the password P, a random salt S, and a conventional symmetric encryption scheme(SE).
- Key K is obtained by hashing the salt and password and returns as ciphertext (S, C) where C $\leftarrow$ SE.Enc(K, M) is encryption of M under K.
- **Limitation of Traditional PBE:** TPBE only provides message privacy under **chosen-plaintext attack**, and it fails to define or prove authenticity.

Chosen-plaintext Attack

- The attacker may choose n plaintexts.
- The attacker then sends these n plaintexts to the encryption oracle.
- The encryption oracle will then encrypt the attacker's plaintexts and send them back to the attacker.
- The attacker receives n ciphertexts back from the oracle, in such a way that the attacker knows which ciphertext corresponds to each plaintext.
- Based on the plaintext–ciphertext pairs, the attacker can attempt to extract the key used by the oracle to encode the plaintexts. Since the attacker in this type of attack is free to craft the plaintext to match his needs, the attack complexity may be reduced.

Flexible Password-Based Encryption (FPBE)

- It has a new syntax and security definition.
- The key(password) can be of any length and the encryption algorithm is deterministic, taking the salt, nonce, and associated data as input. We write C $\leftarrow$ FPBE.Enc(P, S, N, M, A).
- Decryption algorithm requires the salt, nonce, and associated data to be sent out of the band. We write M $\leftarrow$ FPBE.Dec(P, S, N, C, A).

Security features of FPBE

- **Privacy(PIND$):** It requires ciphertext to be indistinguishable from random strings when the salt is honestly chosen, and a nonce is not repeated for a given salt.
- **Authenticity(PAUTH):** This makes it infeasible to produce S, N, C, A(salt, nonce, ciphertext and the associated data) except in a trivial way.
- **PAE:** It captures both privacy and authenticity in a single, integrated way.
  - **Theorem 1:** If FPBE scheme separately satisfies PIND$ and PAUTH, then it also satisfies PAE.

Features of FPBE

- FPBE allows the salt to be securely reused to encrypt multiple messages as long as the nonce is different each time.
- Associated data could be metadata (such as a file handle) and is authenticated but not encrypted.
- Privacy is strengthened compared to TPBE by requiring indistinguishability from random rather than indistinguishability of encryptions, which provides some degree of anonymity.
- The main value added is authenticity, not present in TPBE, and is crucial for modern applications.

Applications

- Securing data stored in the cloud - TBPE is not a good fit for this task because it provides privacy and not authenticity, which is necessary to ensure that the data has not been tampered by a malicious server.
- Reducing storage costs - In TPBE, each message is encrypted using a random salt which adds up to a lot of storage overhead. In FPBE, we use only one random salt which reduces the amount of storage needed.
- FPBE adds support for nonces and associated data and provides both privacy and authenticity, which is consistent with AEAD(Authenticated encryption with associated data).
- Provably resisting **partitioning-oracle attacks**.

Partitioning-Oracle Attacks

- The TPBE scheme uses a conventional symmetric encryption scheme called the base scheme. DtE(Derive then encrypt) FPBE scheme will also use the same scheme.
- A base scheme is considered key-robust (or key-committing) if a ciphertext serves as a commitment to the key. It means that once a ciphertext is generated, it cannot be decrypted with any other key.
- Partitioning-Oracle Attacks can exploit the lack of key robustness in the base scheme to speed up password recovery in the corresponding TPBE scheme.

- The FPBE framework considers attacks that violate authenticity, including partitioning-oracle attacks.
- The presence of key robustness in the base scheme can resist partitioning-oracle attacks using FPBE.
- Assume that the attacker has access to the password's dictionary and the server's password is also present in that dictionary.

Brute-force Attack

- Attacker takes the passwords one by one and sends it to the server after encrypting them. If the server responds with the decryption error then he continues on to the next password.
- This would take O(d) time, where d is the size of the password dictionary.

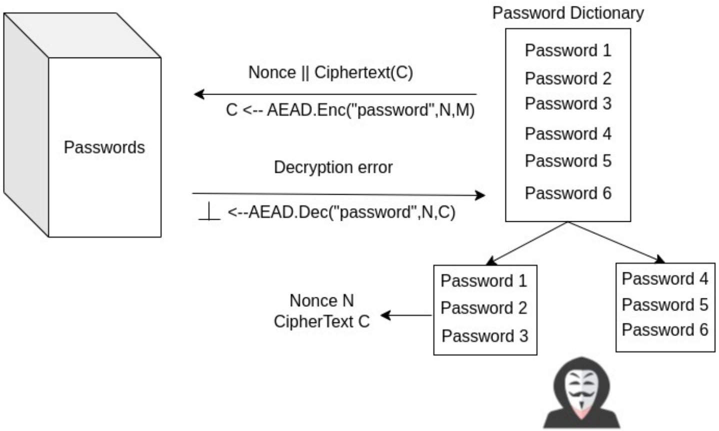

Figure 1: Partitioning-Oracle Attacks

- Attacker splits the passwords into two sets and calculates the nonce of each set in such a way that every password of one set can decrypt the message while other sets cannot.
- Oracle-partitioning attacks eliminate half of the password set each time, thus reducing the complexity to O(log d).

The DtE Scheme

- The DtE is a way to create a secure encryption scheme called FPBE.
- It has two main ingredients:
  - A conventional Authenticated encryption with associated data (AEAD) scheme, SE
  - A password-based key-derivation function(PBKDF), F
- The security of the DtE scheme depends on the strength of the password used.
- The password needs to be strong, which means it should be hard to guess, in order to provide security for the PBE scheme.

Security of DtE scheme

- The security of the DtE scheme can be analyzed at two levels:
  - Asymptotic (or Qualitative)
  - Concrete (or Quantitative)

- At the asymptotic level, assuming the passwords is un-guessable,

  - **Theorem 2:** If base scheme SE provides privacy, then FPBE meets our PIND$, definition of privacy.

  - **Theorem 3:** If base scheme SE provides authenticity, then FPBE meets our PAUTH, definition of authenticity.

  - **Theorem 4:** If base scheme SE provides both authenticity and key-robustness, then FPBE again meets PAUTH.

- At this level, looks like redundancy; why do we add an extra assumption (key-robustness) on SE to obtain the same result as in **theorem 3**.
- At the concrete level, the security of the DtE scheme is analyzed by bounding the advantage of an adversary in violating the privacy or authenticity of FPBE.
- The bound depends on the number of guesses q that the adversary can make to guess the password. The lower the value of q, the better the result.

Salts vs. Nonces

- Salt is used to prevent pre-computation in brute-force attacks, forcing the attacker to do dictionary-size work.
- Salt makes each user's password unique, making it more difficult for an attacker to pre-compute a list of possible passwords.
- Nonces are used to make ciphertexts unique. However, nonces can be predictable and the same for different users, so they do not provide the same level of security as salts.
- Using both salts and nonces can have benefits, such as reducing storage costs in cloud encryption.

Storage Overhead In TPBE

- With traditional PBE, each message is encrypted with a per-message random salt, which needs to be stored along with the ciphertexts. This results in a storage overhead of sl * q (sl is the salt length, q is the number of messages).

- With flexible PBE, the user picks a single random salt S and encrypts each message M with salt S and a nonce $\langle i \rangle c$, where i is the index of the ciphertext in the list and c is a small encoding of i.

- This results in a storage overhead of sl + qc, which is lower than sl * q.

- In fact, with FPBE, only the single salt needs to be stored since the nonce can be derived from the index i of the ciphertext. This results in a storage overhead of just sl.

Security Goals

Some terminologies

'PD'	Password distribution
'UN'	A set of unique nonce
'N'	Nonce
'M'	Message
'C'	Ciphertext
'A'	Additional data
'MT'	Message table
'CT'	Cipher table
'S'	List of salts
'cl'	Cipher length
'Cd'	Combination of C0 and C1
'FPBE.SS'	Generates a new salt

Goal 1: Privacy(PIND$)

- INIT:

```
1. d ←{0,1} ; un ← true
2. P ← PD // u-vector of passwords
```

- ENC(i, N, M, A):

```
1. Require s(i) $\neq 0$
2. Require CT[$i, s(i), N, M, A$] $= \perp$
3. if ($N \in UN_{i,s(i)}$) then un ← $false$
4. UN$_{i,s(i)}$← UN$_{i,s(i)} \cup \{N\}$
5. C$_1$ ← $FPBE.Enc(P[i], S_i, N, M, A)$
6. C$_0$ ← $\{0,1\}^{FPBE.cl(|M|)}$
7. CT[i, s(i), N, M, A] ← C$_d$
8. Return C$_d$
```

- SALT(i):

```
1. s(i) ← $s(i) + 1; S_i \leftarrow FPBE.SS$
2. Return S$_i$
```

Goal 2: Authenticity(PAUTH)

- INIT:

```
1. un ← true;
2. P ← PD // u-vector of passwords
```

- ENC(i, N, M, A)

```
1. Require s(i) $\neq 0$
2. if($N \in UN_{i,s(i)}$) then un ← false
3. $\mathrm{UN}_{i,s(i)} \leftarrow \mathrm{UN}_{i,s(i)} \cup \{N\}$
4. C ← FPBE.Enc(P[i],$\mathrm{S}_i, N, M, A$)
5. MT[i,$\mathrm{S}_i, FPBE.NI(N), C, A$]←M
6. Return C
```

- VERIFY(i, S, I, C, A):

```
1. if($MT[i, S, I, C, A] \neq \perp$) then return $\perp$
2. M ← $FPBE.Dec(P[i], S, I, C, A)$
3. if($M \neq \perp$) then win ← $true$
4. return (M $\neq \perp$)
```

- SALT(i):

```
1. s(i) ← $s(i) + 1; S_i \leftarrow FPBE.SS$
2. Return $\mathrm{S}_i$
```

Goal 3: Privacy+Authenticity(PAE)

- INIT:
  1. d $\leftarrow$\{0,1\} ; un $\leftarrow$ true
  2. P $\leftarrow$ PD // u-vector of passwords
- ENC(i, N, M, A)
  1. Require s(i) $\neq$ 0
  2. Require CT[*i, s(i), N, M, A*] = $\perp$
  3. if ($N \in UN_{i,s(i)}$) then un $\leftarrow$ *false*
  4. $\text{UN}_{i,s(i)} \leftarrow \text{UN}_{i,s(i)} \cup \{N\}$
  5. $\text{C}_1 \leftarrow FPBE.Enc(P[i], S_i, N, M, A)$
  6. $\text{C}_0 \leftarrow \{0,1\}^{FPBE.cl(|M|)}$
  7. CT[i, s(i), N, M, A] $\leftarrow \text{C}_d$
  8. MT[i,$\text{S}_i, FPBE.NI(N), C_d, A$]$\leftarrow$M
  9. Return C_d
- DEC(i, S, I, C, A):
  1. if($MT[i, S, I, C, A] \neq \perp$) then return MT[i,S,I,C,A] $\neq\perp$
  2. M $\leftarrow FPBE.Dec(P[i], S, I, C, A)$
  3. return M
- SALT(i):
  1. s(i) $\leftarrow s(i) + 1; S_i \leftarrow FPBE.SS$
  2. Return S_i

Conclusion

- The paper discusses different ways of defining authenticated encryption, which is a way of encrypting data securely while also ensuring that it hasn't been tampered with.
- It describes how we can use a unique nonce while encryption through a encryption scheme called FPBE to attain both privacy and authenticity.
- It also discusses how we can resist partitioning oracle attacks with the help of above mentioned scheme.

References

[1.] J. Len, P. Grubbs, and T. Ristenpart. Partitioning oracle attacks. In M. Bailey and R. Greenstadt, editors, 30th USENIX Security Symposium. USENIX Association, 2021.

[2.] B. Kaliski. PKCS 5: Password-Based Cryptography Specification Version 2.0. RFC 2898, Sep. 2000.

[3.] M. Bellare, R. Ng, and B. Tackmann. Nonces are noticed: AEAD revisited. In A. Boldyreva and D. Micciancio, editors, CRYPTO 2019, Part I, volume 11692 of LNCS, pages 235–265. Springer, Heidelberg, Aug. 2019.

[4.] Boxcryptor. Technical overview. https://www.boxcryptor.com/en/technical-overview/, visited on October 17, 2022.

[5.] M. Bellare, R. Ng, and B. Tackmann. Nonces are noticed: AEAD revisited. In A. Boldyreva and D. Micciancio, editors, CRYPTO 2019, Part I, volume 11692 of LNCS, pages 235–265. Springer, Heidelberg, Aug. 2019.

Thank You !!!