No Key, No Problem:

Vulnerabilities in Master Lock Smart Locks

Chengsong Diao*, Danielle Dang*, Sierra Lira*, Angela Tsai*,
Miro Haller*, Nadia Heninger*

*UC San Diego

USENIX WOOT; August 11, 2025

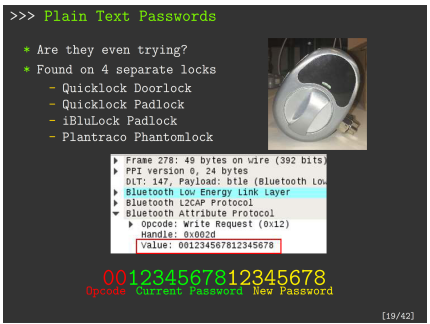
Introduction



Charles Curi

- Popular: $2.8 billion global market size
- Security critical

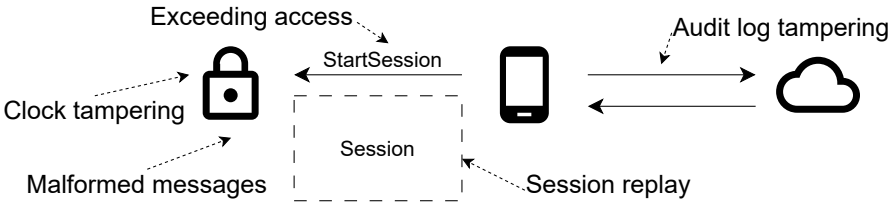
Security

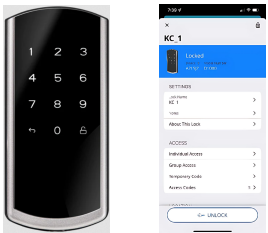


DEF CON 2016 Rose and Ramsey

- Previous studies found vulnerabilities in August, Dana, Kevo, ...
- Passwords in plaintext, replay attacks, etc.
- Rose and Ramsey (DEF CON 2016): Master Lock not vulnerable
- Knight, Lord, and Arief (IEEE TrustCom 2019): stopped at encrypted communication

Contributions



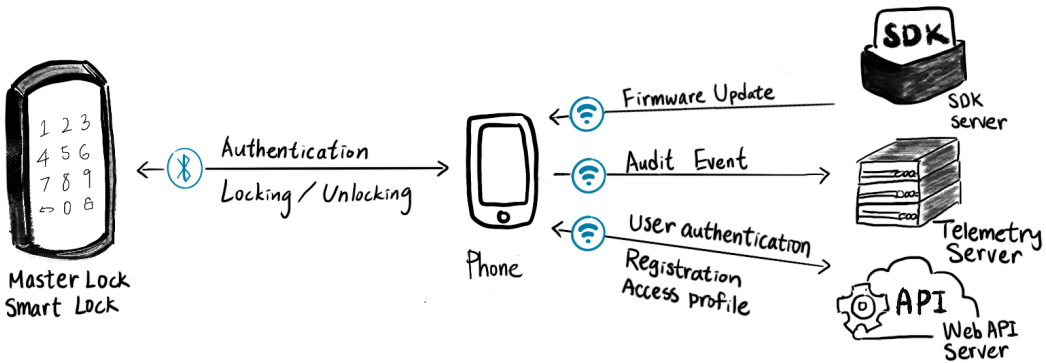


Master Lock D1000 and Enterprise app

- Reverse engineered the Master Lock D1000 and the Master Lock Vault Enterprise Android app
- Found five attacks, including:
  1. **Exceeding access**: unlock after access is revoked
  2. **Session replay**: replay recorded BLE sessions to unlock
  3. **Audit log tampering**: forge audit events
  4. Clock Tampering: maliciously extend access
  5. Malformed Messages: memory safety vulnerability

Background

Architecture



- Device Gateway Architecture: smart lock does not connect to Wi-Fi
- Smart lock communicates with server via phone
- We analyzed BLE communication between lock and phone
- We analyzed HTTPS communication between phone and servers

Bluetooth Low Energy

- Lock only transmits after touchpad is tapped
- BLE transmissions follow GATT protocol
  - Service UUID and device ID for scanning and connecting
  - Characteristic UUID for reading and writing data
- We reverse engineered the app to figure out service and characteristic UUIDs
- We need to understand this protocol to write our custom client



Master Lock D1000 Touchpad

Reverse Engineering

Reverse engineering

- Android app: Master Lock Vault Enterprise, version 2.28.0.1
- Obfuscation techniques: renaming, **encoded and encrypted constants**, **calling native functions**, using reflection, **loading DEX dynamically**, obfuscating control flows

```
1       Object r43 = Class.forName($$c((byte) (-r42[270(0x10e, float:3.78E-43)]),
↪     r42[946(0x3b2, float:1.326E-42)], (short) 866)).getMethod($$c(r42[109(0x6d,
↪     float:1.53E-43)], r42[426(0x1aa, float:5.97E-43)], (short) (-r42[40])), new
↪     Class[0]).invoke(null, null);     // Catch: Exception -> L967
2       if (r43 != null) goto L970;
3     L1762:
4       r43 = Class.forName($$c((byte) (-getLocation[270(0x10e, float:3.78E-43)]),
↪     r10[430(0x1ae, float:6.03E-43)], (short) 524)).getMethod($$c(r10[10], r10[388(0x184,
↪     float:5.44E-43)], (short) 996), new Class[0]).invoke(null, null);     // Catch:
↪     Exception -> L1687
5     L970:
6       if (r43 != null) goto L1788;
```

Example of obfuscated code

Obfuscation technique: encoded and encrypted constants

```
1       Object[] objArr = new Object[1];
2       $$a((char) ((SystemClock.uptimeMillis() > 0L ? 1 : (SystemClock.uptimeMillis() == 0L
    ? 0 : -1)) - 1), 617 - (ViewConfiguration.getMaximumDrawingCacheSize() >> 24),
    (PointF.length(0.0f, 0.0f) > 0.0f ? 1 : (PointF.length(0.0f, 0.0f) == 0.0f ? 0 : -1)) +
    50, objArr);
3       BigInteger bigInteger = new BigInteger(((String) objArr[0]).intern());
4       Object[] objArr2 = new Object[1];
5       $$a((char) (Color.blue(0) + 15189), (SystemClock.currentThreadTimeMillis() > (-1L) ?
    1 : (SystemClock.currentThreadTimeMillis() == (-1L) ? 0 : -1)) + 616, 666 -
    ExpandableListView.getPackedPositionChild(0L), objArr2);
6       try {
7           RSAPrivateKeySpec rSAPrivateKeySpec = new RSAPrivateKeySpec(new
    BigInteger(((String) objArr2[0]).intern()), bigInteger);
```

Obfuscated code to decode static RSA private key

- Constants are stored encoded and encrypted, and are decoded / decrypted at run-time
- Solution: dynamic partial evaluation

Obfuscation technique: calling native functions

```
1       Object[] objArr = new Object[1];
2       $$a((char) ((SystemClock.uptimeMillis() > 0L ? 1 : (SystemClock.uptimeMillis() == 0L
    ? 0 : -1)) - 1), 617 - (ViewConfiguration.getMaximumDrawingCacheSize() >> 24),
    (PointF.length(0.0f, 0.0f) > 0.0f ? 1 : (PointF.length(0.0f, 0.0f) == 0.0f ? 0 : -1)) +
    50, objArr);
3       BigInteger bigInteger = new BigInteger(((String) objArr[0]).intern());
4       Object[] objArr2 = new Object[1];
5       $$a((char) (Color.blue(0) + 15189), (SystemClock.currentThreadTimeMillis() > (-1L) ?
    1 : (SystemClock.currentThreadTimeMillis() == (-1L) ? 0 : -1)) + 616, 666 -
    ExpandableListView.getPackedPositionChild(0L), objArr2);
6       try {
7           RSAPrivateKeySpec rSAPrivateKeySpec = new RSAPrivateKeySpec(new
    BigInteger(((String) objArr2[0]).intern()), bigInteger);
```

Obfuscated code to decode static RSA private key

- Constants are replaced with native function calls and fix-up calculation
- Solution: replace native functions with placeholders

Obfuscation technique: loading DEX dynamically

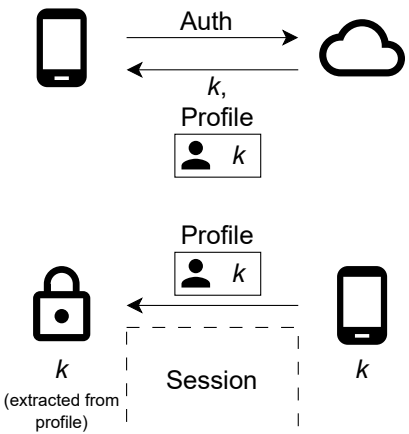
```
1        ZipInputStream r350 = new ZipInputStream(r413);     // Catch: Throwable -> L1538
2        ZipEntry r433 = r350.getNextEntry();     // Catch: Throwable -> L1538
3        int r534 = stopSDK + 77;
4        stopScanning = r534 % 128;
5        int r535 = r534 % 2;
6        r97 = r810;
7        Object r351 = Class.forName($$c(r220[100], (byte) (-r220[704(0x2c0,
↪    float:9.87E-43)]), r220[46])).getDeclaredConstructor(new
↪    Class[]{Class.forName($$c(r220[100], r220[6], r97))}).newInstance(new Object[]{r350});
↪      // Catch: Throwable -> L1586
```

Obfuscated code to read ZIP archive decrypted from APK assets

- Critical code is stored in encrypted assets and loaded at run-time
- Solution: write custom DEX loader and convert to Java libraries

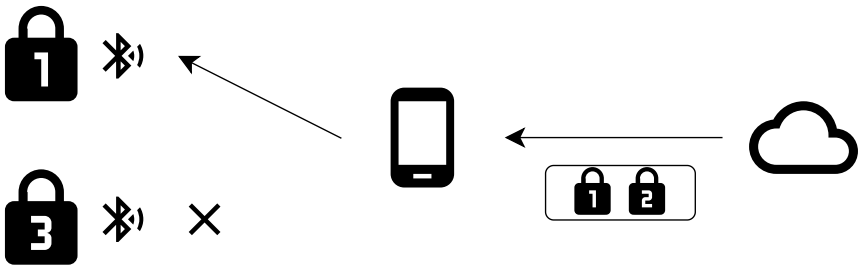
Protocols and Attacks

Authentication flow



1. App authenticates to server using user's credentials
2. Server sends a pair of session key k and profile to app
3. App transmits profile to lock to start a session

Revocation



- App fetches a list of locks to which user has access from server
- App will not show locks not on the list, even if they are discovered in BLE scans
- If administrator revokes user's access, the lock is removed from the list sent by server
- Server will refuse to generate session keys and profiles if user no longer has access

Exceeding access attack

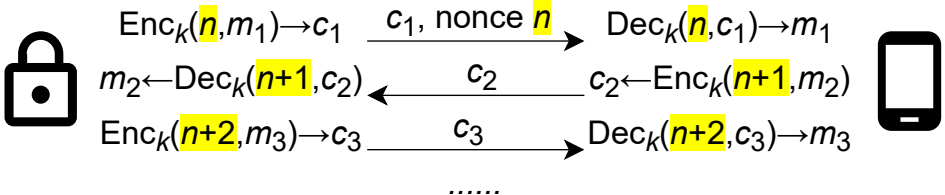
Targeting phone-lock communication

```
{
    "profiles": [
        {
            "id": 128602,
            "accessProfile": "xnd+jiDJpP587jpvhoHT4yS0GU3fnYlGYM5LzIrFQH8=|gLaPSy/n ...
            "startsOn": "1970-01-01T00:00:00+00:00",
            "expiresOn": "2025-08-02T02:15:17.1199254+00:00",
            "refreshThresholdMinutes": 387360
        }
    ]
}
```

Example of device profile returned by server

- Attackers can save session keys and profiles and use them with custom clients, or use official app with phone offline
- Lock can only verify profile validity by expiration time encoded in profiles
- Impact: users can open lock up to **9 months** after access revocation

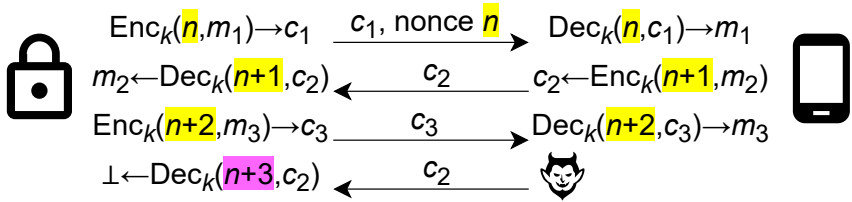
Encryption in BLE communications



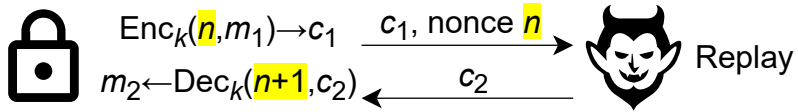
- BLE communication is encrypted using AES-CCM with session key k

Session replay attack

Targeting phone-lock communication



- Incrementing nonce protects against message replay within session

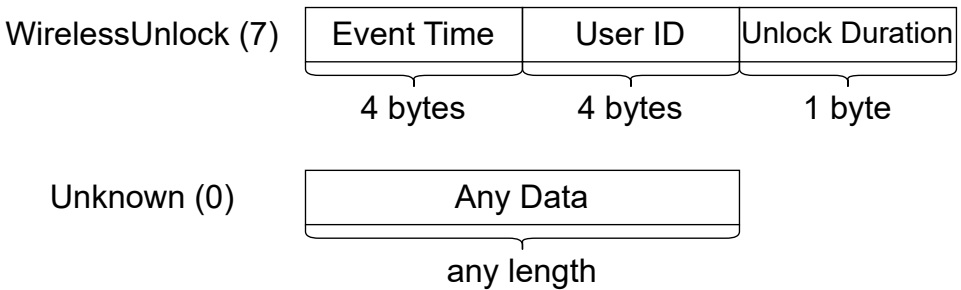


- Lock picks the same starting nonce for every session
- Attackers can record whole session and replay

Audit log tampering attack

Targeting phone-server communication

- App uses static secrets to authenticate to audit log server
- Log entries are not encrypted or authenticated
- We reverse engineered the static secrets and binary formats of log entries
- Attackers can forge arbitrary audit log entries on the server



Disclosure and Mitigation

Disclosure

- We disclosed all attacks to Master Lock
- Master Lock discussed every vulnerability with us
- We appreciate their prompt and detailed response

Mitigation for exceeding access attack

- Max profile expiration date must balance functionality and security
- Longer validity period supports offline unlocking
  - E.g., a cabin in the forest
- However, for D1000, validity period is **9 months**
- This is due to a firmware bug that caused the clock to jump forward in time
- Mitigation: Master Lock has shortened the validity period to one week

Mitigation for session replay attack

- Master Lock confirmed that it was a firmware bug specific to D1000
  - Master Lock has a fix in the next firmware update
- Other Master Lock products choose random starting nonce

Mitigation for audit log tampering attack

- Our recommended mitigation: add MACs to log entries
- After some discussions, Master Lock told us that they have fully remediated this attack
  - We have no further technical details

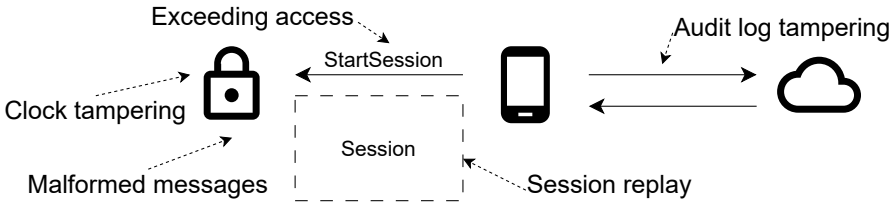
Conclusion

What can we learn?

- Encryption alone does not ensure security
- Open problem: secure channels between IoT devices and remote servers without trusted intermediaries
  - Matter is a standard on secure integration of IoT devices over different protocols, but it requires trusted components to provision devices and issue certificates
  - Master Lock needs secure channels across IP (server-to-phone) and BLE (phone-to-lock), but a user's phone cannot always be trusted



Conclusion



- We discovered five attacks on the Master Lock D1000, including exceeding access, session replay, and audit log tampering
- Details and other attacks can be found in our paper: **Chengsong Diao**, Danielle Dang, Sierra Lira, Angela Tsai, Miro Haller, Nadia Heninger. *No Key, No Problem: Vulnerabilities in Master Lock Smart Locks*. WOOT'25.

QR code to paper.