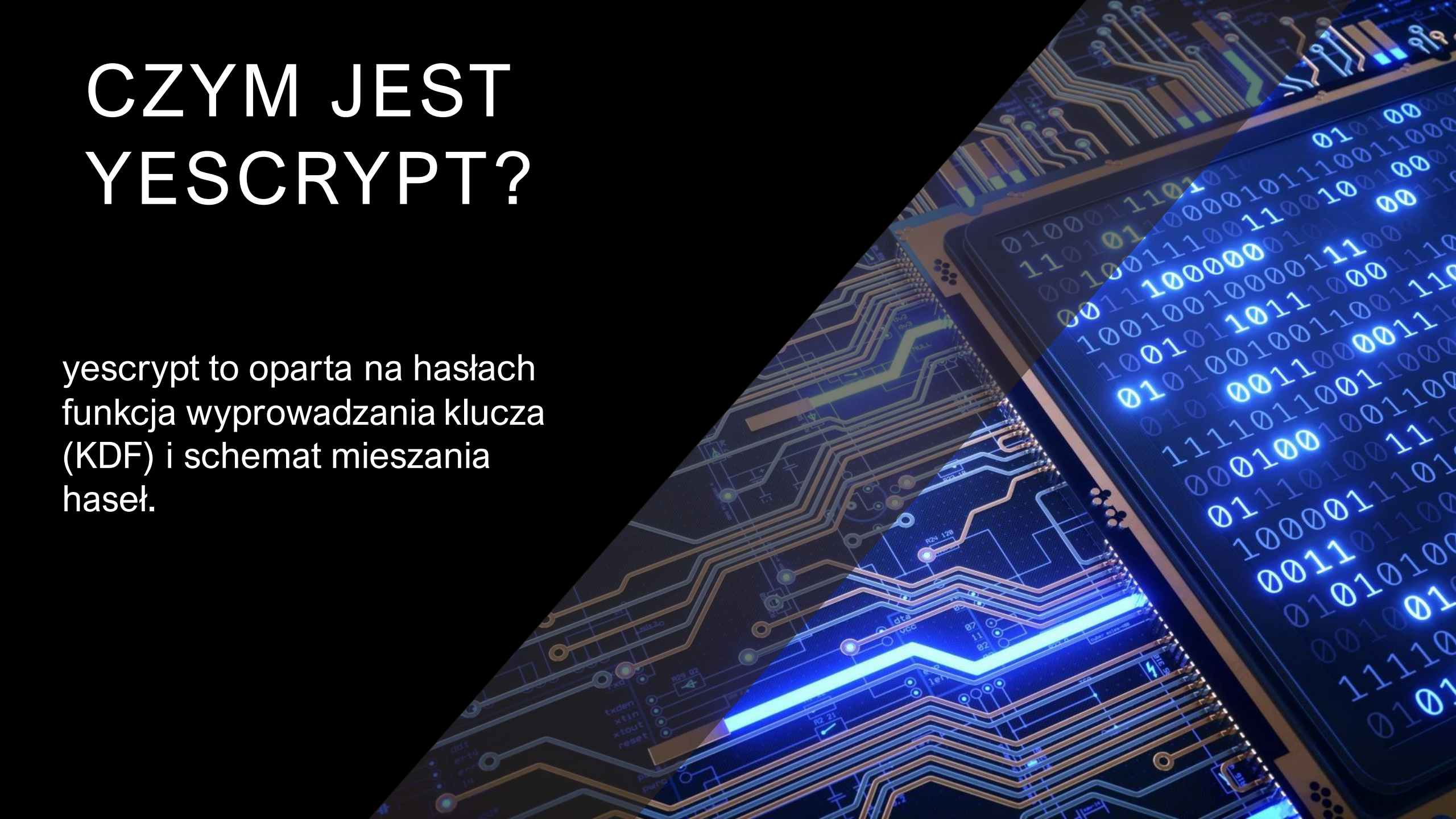
YESCRYPT

Rafał Dadura, Juliusz Kuzyka

CZYM JEST YESCRYPT?

yescrypt to oparta na hasłach funkcja wyprowadzania klucza (KDF) i schemat mieszania haseł.



Dlaczego yescrypt?

- yescrypt adresuje wady takie jak niskie zużycie pamięci, poprzez opcjonalne inicjalizowanie i ponowne wykorzystanie dużej tabeli przeglądowej, co utrudnia atakującym wykorzystanie sprzętu botnetów.
- yescrypt wprowadza inne zmiany, które spowalniają procesory graficzne, układy FPGA i ASIC nawet przy niskim zużyciu pamięci, oraz oferuje dodatkowe funkcje i ustawienia.
- yescrypt jest najbardziej skalowalnym schematem mieszania haseł, zapewniającym wysoki poziom bezpieczeństwa od ataków łamania haseł offline w zakresie od kilobajtów do terabajtów i więcej.
- yescrypt jest dostępny dla różnych systemów operacyjnych i jest zalecany jako domyślny schemat mieszania haseł w niektórych dystrybucjach Linuxa.

Porównanie do scrypt i Argon2 - zalety

- Większa odporność na ataki offline (zwiększenie kosztu atakującego przy takim samym koszcie obrońcy).
- Dodatkowe opcjonalne wbudowane funkcje, np. szyfrowanie skrótu, aby skrótów nie można było złamać bez klucza (do przechowywania osobno).
- Bezpieczeństwo kryptograficzne zapewniane przez prymitywy zatwierdzone przez NIST.
- SHA-256, HMAC, PBKDF2 i scrypt są użyteczne z tej samej bazy kodu.

Porównanie do scrypt i Argon2 - wady

- Złożony (większe ryzyko wystąpienia błędu ludzkiego i pozostania niezauważonym przez długi czas)
- Niebezpieczne synchronizowanie pamięci podręcznej (jak bcrypt, scrypt i Argon2d, ale w przeciwieństwie do Argon2i)
- Nie był zwycięzcą PHC (był nim Argon2), ale finalista ze „specjalnym uznaniem”
- Obsługiwane w mniejszej liczbie projektów innych firm (libxcrypt , Linux-PAM , shadow , mkpasswd)

Kod

- Protokół yescrypt składa się z 3 części:

1. Inicjalizacja
2. Pętla Scrypt
3. Zwrócenie wyniku

Inicjalizacja

```
// Initialization
MessageDigest sha256Digest = MessageDigest.getInstance( algorithm: "SHA-256");
sha256Digest.update(password);
sha256Digest.update(salt);
byte[] blockhash = sha256Digest.digest();
```

Pętla Scrypt

1. Mieszanie bloku
2. Wewnętrzna pętla
3. XORowanie
4. Mieszanie bufora
5. Zapis bloku
6. Sprawdzenie warunku zakończenia

Mieszanie bloku

- Początkowy blok mieszania jest mieszany z hasłem, solą i skrótem poprzedniego bloku. Wykorzystywane są operacje XOR i funkcje skrótu.

```
// Mix blockhash with password and salt
sha256Digest.reset();
sha256Digest.update(blockhash);
sha256Digest.update(password);
sha256Digest.update(salt);
byte[] x = sha256Digest.digest();
```

Wewnętrzna pętla

- Wykonuje obliczenia wewnętrzne, które składają się z wielokrotnego mieszania bloku za pomocą funkcji skrótu. Liczba iteracji wewnętrznej pętli zależy od parametru p.

```
// Compute inner loop
for (int j = 0; j < p; j++) {
    sha256Digest.reset();
    sha256Digest.update(x);
    x = sha256Digest.digest();
}
```

XORowanie

- Wynik wewnętrznej pętli jest XORowany z buforem.

```
for (int j = 0; j < r; j++) {
    int startPos = j * 32;
    for (int k = 0; k < 32; k++) {
        if (startPos + k >= buf.length)
            break;
        buf[startPos + k] ^= x[k];
    }
}
```

Mieszanie bufora

- Zaktualizowany bufor jest ponownie mieszany za pomocą funkcji skrótu.

```
// Mix buf
sha256Digest.reset();
sha256Digest.update(buf);
blockhash = sha256Digest.digest();
```

Zapis bloku
Sprawdzenie warunku zakończenia

- Blok wynikowy jest zapisywany w wyjściowej tablicy wynikowej (dk).
- Sprawdzany jest warunek zakończenia pętli, który zależy od rozmiaru wyjściowej tablicy wynikowej.

```
// Write block to dk
int blockLen = Math.min(128, dkLen - dkIndex);
System.arraycopy(buf,  srcPos: 0, dk, dkIndex, blockLen);
dkIndex += blockLen;

if (dkIndex >= dkLen) {
    break;
}
```

bytesToHex

```
2 usages
private static final char[] HEX_ARRAY = "0123456789ABCDEF".toCharArray();

1 usage
public static String bytesToHex(byte[] bytes) {
    char[] hexChars = new char[bytes.length * 2];
    for (int i = 0; i < bytes.length; i++) {
        int value = bytes[i] & 0xFF;
        hexChars[i * 2] = HEX_ARRAY[value >>> 4];
        hexChars[i * 2 + 1] = HEX_ARRAY[value & 0x0F];
    }
    return new String(hexChars);
}
```

Testy

- TestYescrypt – Sprawdza działanie kodu
- TestYescryptWithEmptyPassword - Sprawdza działanie kodu z brakiem hasła
- TestYescryptWithEmptySalt - Sprawdza działanie kodu z brakiem soli
- TestYescryptWithZeroParameters - Sprawdza działanie kodu z brakiem wszystkich parametrów

TestYescrypt

```java
@Test
public void testYescrypt() throws NoSuchAlgorithmException {
    byte[] password = "password".getBytes();
    byte[] salt = "salt".getBytes();
    int N = 16384;
    int r = 8;
    int p = 1;
    int dkLen = 64;

    String expectedKey = "87B2DDE8176361CFE8639B5A9F3E00C14D122E5C652B5D07066F775372DCD58F87B2DDE8176361CFE8639B5A9F
    byte[] result = Yescrypt.yescrypt(password, salt, N, r, p, dkLen);

    Assert.assertEquals(expectedKey, Yescrypt.bytesToHex(result));
}
```

TestYescryptWithEmptyPassword

```java
@Test
public void testYescryptWithEmptyPassword() throws NoSuchAlgorithmException {
    byte[] password = new byte[0];
    byte[] salt = "salt".getBytes();
    int N = 16384;
    int r = 8;
    int p = 1;
    int dkLen = 64;

    String expectedKey = "6BB9A61670F109057586998C72B338B9350572B8654E431CF58E83AFBBE6189A6BB9A61670F109057586998C72
    byte[] result = Yescrypt.yescrypt(password, salt, N, r, p, dkLen);

    Assert.assertEquals(expectedKey, Yescrypt.bytesToHex(result));
}
```

TestYescryptWithEmptySalt

```java
@Test
public void testYescryptWithEmptySalt() throws NoSuchAlgorithmException {
    byte[] password = "password".getBytes();
    byte[] salt = new byte[0];
    int N = 16384;
    int r = 8;
    int p = 1;
    int dkLen = 64;
    String expectedKey = "D01AECC4AE9A0F2F948CCF77E4D9025F3031E6E306A7F6AC4A0D31BFF5610B15D01AECC4AE9A0F2F948CCF77E4
    byte[] result = Yescrypt.yescrypt(password, salt, N, r, p, dkLen);

    Assert.assertEquals(expectedKey, Yescrypt.bytesToHex(result));
}
```

TestYescryptWithZeroParameters

```java
@Test
public void testYescryptWithZeroParameters() throws NoSuchAlgorithmException {
    byte[] password = "password".getBytes();
    byte[] salt = "salt".getBytes();
    int N = 0;
    int r = 0;
    int p = 0;
    int dkLen = 0;

    byte[] expectedKey = new byte[0];
    byte[] result = Yescrypt.yescrypt(password, salt, N, r, p, dkLen);

    Assert.assertArrayEquals(expectedKey, result);
}
```

Bibliografia

- [1] Openwall, "yescrypt," Openwall. [Online]. Available: https://www.openwall.com/yescrypt/}. [Accessed: June 15, 2023].