

Format description: v1.0

Header:

The header consists of a magic number, a version number, some padding, and an initialization vector. All in one contiguous block of bytes.

The magic number consists of 8 bytes, the first four bytes are 0x42, 0x48, 0x50 and 0x4D, which is the ASCII representation of the string "BHPM". These are then followed by the 32-bit integer 0x11223344.

Immediately following the magic number comes a single byte representing the major version of the file format, followed by a padding byte that should be all zeros. After that is another byte representing the minor version, and a padding byte with only zeros.

Finally at the end of the header comes the initialization vector for the AES-128 encryption. It consists of 16 bytes.

The header may be extended in future versions.

Decryption:

The rest of the file can be decrypted with AES-128, and it shall be configured as follows: Use the initialization vector stored in the header and derive the key from a password using SHA-256. The block cipher mode is CBC, with no padding.

Once decrypted the result should be a sequence of bytes with a length that is a multiple of 16 bytes. After this step the file is still not quite readable, and it is not yet possible to verify if it was decrypted correctly.

To fully uncover the cleartext the bitwise XOR operation must be applied to the output of the decryption step using bytes derived from the xorshift128+ algorithm, with the seed being the last 16 bytes of the output.

Xorshift128+ algorithm in C:

```
uint64_t state[2]; //Must be seeded
uint64_t xorshift128plus(void)
{
    uint64_t x = state[0];
    uint64_t y = state[1];
    state[0] = y;
    x ^= (x << 23);
    state[1] = x ^ y ^ (x >> 17) ^
                (y >> 26);
    return state[1] + y;
}
```

Decryption step using xorshift128+:

For every 64-bit chunk in the output, skipping the last 16 bytes, generate a 64-bit value using xorshift128+ and perform the bitwise XOR operation between the chunk and the value, and overwrite the chunk with the result.

After this step, the result can be verified by comparing the first 32 bytes in the output with the SHA-256 hash of rest of the output, including the 16 bytes of seed. If they match, the file was decrypted correctly, and the content can be extracted by skipping the first 32 bytes and the last 16 bytes of the output.

Content:

The main content of the file has a simple format. It starts with a 32-bit integer which represents the number of "entries" in the file. It is followed by another 32-bit integer containing the total size of the content, which may be smaller than the total number of bytes extracted. If it is smaller, then the content was padded with random bits at the end.

Following this content header comes the actual data in blocks of variable length. In order to decode these blocks the data must be parsed sequentially. The block begins with a single byte representing the length of the id and another byte representing the length of the password, which is then immediately followed by the UTF-8 encoded id and password. If it is not the final block then another block follows immediately after.

Notes:

All integers are unsigned unless noted otherwise. Any integer stored as more than one byte is stored using little-endian byte order. That is, the sequence 0x20, 0x40, 0x60, 0x80, shall be interpreted as the 32-bit unsigned integer 0x80604020 which in decimal is represented as "2153791520".