Moneybelt litepaper v0.5

Edward Keyes <ed@attocash.com>
Last modified: 2023-10-05

An authenticated-encryption protocol designed for Atto¢ash tickets

Introduction

Atto¢ash is a micropayment system under development which uses digital-cash **tickets** to represent small quantities of money that can be passed from an issuer to a recipient. These tickets need to be highly secure, both to ensure the privacy of the users and to prevent the possibility of tampering with the data (turning “$0.01” into “$100”) or outright counterfeiting.

Cryptography is difficult and subtle, and its history is filled with seemingly clever schemes that were later shown to be fatally flawed. To avoid this fate, we need to base our solution on proven standards. However, our scenario is somewhat unusual, and in particular we want to keep the tickets as short as possible so that they can be easily sent as clickable URLs in texts, chats, emails, QR codes, etc. For example:

https://attocash.com/USD/0.01/2Wvr9E31kywdLzpxRQXaFZhoCJtQWWGV

Given this constraint, many existing options such as public-key digital signatures are not a good match. However, we also have some unusual advantages that we can leverage to pick the right standard technologies and mix them together in the right way.

The result is Moneybelt, a specification for how Atto¢ash tickets are constructed and validated. Its name is the answer to a question: if you’re bringing small quantities of cash from one place to another, where do you keep it that’s more secure than your wallet or your pocket?

Features

Moneybelt provides several advantages to recommend it:

- Secure encryption for the private ticket contents using **AES**
- Authentication for both the visible and private ticket contents using **SHA3**
- Several optimizations for short messages:
    - Variable-length signatures of 0 to 32 bytes
    - Encrypted size is just one byte larger than the plaintext plus signature
    - Minimum total size of just one AES block of 16 bytes

However, it also has some disadvantages to take into account:

- Trivially vulnerable to replay attacks: identical messages encrypt identically
- Full strength relies on some uniqueness properties of the plaintext
- Authentication is not public: only the user or the server can validate a message
- Not optimized for high-speed performance: two hashes and two encryptions
- Designed by a software engineer, not a professional cryptographer!

The uniqueness property above deserves some explanation. Many classic ciphers use an **initialization vector** (IV) to start off the encryption in an unpredictable state. This prevents wide classes of attacks and ensures that if messages are repeated, their ciphertext will be different instead of revealing that the same one was sent again. However, the IV is typically a whole block which must be sent with the data, a substantial overhead for short messages.

On the other hand, Atto¢ash tickets by design are unique, as part of the protection against double spending, so we know they will never be repeated. In this way we can avoid the need for an IV at all while still retaining full security. Just bear this in mind if you are considering adopting Moneybelt for other applications. If an IV is needed, the protocol is specified with that option too.

Moneybelt uses symmetric rather than public-key cryptography, so knowledge of the secret key is required to encrypt, decrypt, sign, or verify. It would have been somewhat useful for tickets to be also verifiable offline by the recipient. However, even if the ticket is correct, it still needs to be presented to the Atto¢ash server to be claimed, and at that point it could easily be discovered that it has already been spent, despite the signature, so public verification isn't that important.

Authentication

To create its **message authentication code** (MAC), Moneybelt relies on the industry-standard HMAC construction, using the SHA3-256 hash algorithm. For more details see the Wikipedia article on the technique, but it is a two-pass process which hashes both the plaintext and the secret key together so that only a key-holder can create or verify the MAC:

pass 1: $temp = H(\ (K + pad_1)\ |\ M\ |\ A\ )$
pass 2: $MAC = H(\ (K + pad_2)\ |\ temp\ )$

Here H is the hash function, K is the key, M is the plaintext message, A is any associated data to also be authenticated (such as the non-encrypted portions of the ticket), and the two padding values are XORed with the key and extend it to a full block size if necessary. In Moneybelt, M and A are also prefixed with a **varint** encoding of their lengths (from Google protocol buffers) to avoid ambiguity in the concatenation.

Once we have the full 32-byte MAC, we have the option to truncate it. The default 8-byte MAC, for example, will provide a 2^{-64} (about 1 part in 10^{19}) probability of false validation. In Atto¢ash, the strength can vary with the value of the ticket and user preferences, since it can be overkill to protect a ticket worth 1/10 of a cent with military-strength authentication.

To truncate the MAC to *N* bytes (range of 0-32), we just take the first *N* bytes of the hash output and then append a final length byte whose lower 5 bits contain *N*. The upper 3 bits contain the next 3 (most significant) bits of the hash as a bonus, so *N*=1 actually provides 2^{11} authentication strength instead of 2^8. For *N*=31 (the upper limit of 5 bits), we treat that as *N=32* to provide the full hash, and the extra 3 bits in the length byte are set to 0 and reserved for future options like using SHA3-512 for even longer MACs.

If the size of the message plus truncated MAC is less than one AES block of 16 bytes, then we just increase *N* to pad with additional hash bits "for free". In the special case of *N*=0, indicating no MAC is desired, we instead just append some standard padding composed of a single 1 bit in the first, most significant position followed by zeros, which would look like a hexadecimal sequence of `80` or `80 00 00`, etc. For consistency in the *N*=0 case, the length byte is given as `80` even in the case of messages longer than one block where no padding is needed.

Encryption

The payload to be encrypted is created by concatenating the plaintext message with the truncated MAC and its length byte. The overall size is thus equal to the length of the message plus the length of the MAC plus 1 byte, which is the theoretical minimum to contain the necessary information. Although the MAC does not actually need to be encrypted, messages smaller than one block will benefit from having the hash bytes to pad the block.

Moneybelt uses the industry-standard AES encryption algorithm, with any of the defined key sizes of 128, 192, or 256 bits. The main choice is picking the optimal block-chaining mode. In keeping with minimizing the encrypted size, we want to target modes that can have a partial block at the end, as well as those that are still secure without a separate IV.

Another angle to consider is error propagation. Chaining modes used as stream ciphers often allow bit-flips in the ciphertext to translate directly into bit-flips in the plaintext, particularly in the final block. That sort of behavior still matters even with a MAC, as there is some probability of a false positive with small *N*. In a good case altering the ciphertext produces garbage data which is unlikely to be parsed. But it's a worse compromise if a MAC false positive allows an attacker to use bit-flips to craft a valid malicious ticket and surgically edit the value or recipient.

Moneybelt uses **propagating cipher block chaining** (PCBC) mode. For more details see Wikipedia, but it is a close variant of the standard CBC mode with an extra feed-forward plaintext step. Each ciphertext block is an encryption of the XOR of the current plaintext block, the previous plaintext block, and the previous ciphertext block:

$$C_i = E(P_i + P_{i-1} + C_{i-1})$$

This amplifies the error propagation in subsequent blocks rather than re-syncing after an error. The mode is also amenable to adding **ciphertext stealing** (CTS), where a final partial block is entangled with the second-to-last block in order to be securely encoded without any extra bytes: see Wikipedia. The only difference in PCBC mode is that the final plaintext block is padded with bytes from the previous plaintext block instead of with zeros, to match the feed-forward step.

The error propagation is still not perfect, however, since full scrambling only applies to blocks after a bit-flip occurs, and in short messages there may not be such a block. So Moneybelt takes care of that by performing two-pass encryption. After the first normal pass, all of the ciphertext bytes are reversed in order and encrypted again with the same key. In this way, errors propagate both forwards and backwards, so a bit-flip anywhere in the ciphertext should result in essentially a complete randomization of the entire decrypted plaintext, disallowing an attacker any control over what specific edits they can perform, even before the MAC is considered.

As mentioned previously, the PCBC-CTS cipher mode in Moneybelt is specified with the option for an IV, which is applied at the start of encryption in both directions. However, Atto¢ash itself uses an all-zero IV, which doesn't need to be included in the ticket. With this option, the first block of the ciphertext is just a straight AES encryption of the first block of plaintext. In making this choice, we are relying on the uniqueness of the first part of the ticket data, and have included safeguards in the ticket-generation libraries to ensure this condition is always met.

That being said, the two-pass PCBC encryption does gain us some amount of security leeway. Even if the first block of the plaintext is identical between two messages, as long as they differ by one bit in some other block, those changes will propagate to the end of the first-pass ciphertext, and then back to encompass every block in the reverse pass. So while security is enhanced by first-block uniqueness, including the MAC in the payload and using two passes provide some additional failsafes by amplifying the differences between messages.

API

The reference implementation of Moneybelt is a Python library with a simple API based on the Pycryptodome project, which it uses for the underlying cryptographic operations.

```
class Moneybelt:

  MAC_SIZE = 8

  def __init__(self, aes_key, *, iv=None, mac_size=MAC_SIZE):
    self.aes_key = aes_key
    self.iv=iv
    self.mac_size = mac_size
    self.last_mac_size = None
```

The `Moneybelt` object is constructed by providing an AES key, as a `bytes`-like object of length 16, 24, or 32. The initialization vector of length 16 bytes is optional and will default to all zeroes if not provided. The `mac_size` parameter controls the baseline MAC length used for subsequent operations, but it can be overridden for individual messages.

With this object you can perform encryption and decryption operations, which also include the creation and verification of the MAC.

```
    def encrypt(self, plaintext, extra_data=None, *, mac_size=None)

    def decrypt(self, ciphertext, extra_data=None, *, mac_size=None)
```

Here `extra_data` is some optional associated `bytes`-like data which will be authenticated by the MAC but not encrypted, such as the human-readable part of a ticket URL. If provided at encryption, the identical value must also be provided at decryption for authentication to succeed. The `encrypt()` call will return the ciphertext, and the `decrypt()` call will return the plaintext or `None` if the authentication fails.

On encryption, `mac_size` is treated as a minimum: more MAC bytes than requested might be included as padding for short messages. With a value of 0, the MAC will not be used (or even computed) and only zero padding bytes might be added instead. On decryption, `mac_size` is treated as a minimum requirement: authentication will succeed if the MAC in the ciphertext is the given length or longer and passes verification. A value of 0 will succeed even if the MAC is not included in the ciphertext, but if it is included, it must be valid.

The object also has a `last_mac_size` member variable which records what the actual MAC length was on the last operation. This allows `decrypt()` to potentially be called with a small `mac_size`, and then the caller could choose to accept or reject individual messages based on their decrypted contents afterwards: higher-value tickets might require more secure MACs, so the actual MAC size present in the ticket can be inspected for that check.

Contact

Although based on industry-standard cryptography techniques, Moneybelt is admittedly a new protocol, and previous iterations of the design were found to have some weaknesses. We believe that those problems have been understood and addressed in the current iteration, but any comments, criticisms, and suggestions for existing solutions would be very welcome.

Please reach out to us at: *feedback@attocash.com*