User-friendly Manual Transfer of Authenticated Online Banking Transaction Data
A Case Study that Applies the What You Enter Is What You Sign Transaction Authorization Information Scheme

Sven Kiljan[1,2,3], Harald Vranken[1,3] and Marko van Eekelen[1,3]
[1]*Faculty of Management, Science & Technology - Department of Computer Science, Open Universiteit, P. O. Box 2960, 6401DL Heerlen, The Netherlands*
[2]*Economy & Management - Lectureship Cybersafety, NHL Hogeschool, P. O. Box 1080, 8900CB Leeuwarden, The Netherlands*
[3]*Faculty of Science - Digital Security, Radboud University, P. O. Box 9010, 6500GL Nijmegen, The Netherlands*

Keywords: Online Banking, Security, Authentication, Trusted, Information, Transfer, Human Interaction.

Abstract: Online banking relies on user-owned home computers and mobile devices, all vulnerable to man-in-the-middle attacks which are used to steal money from bank accounts. Banks mitigate this by letting users verify information that originates from these untrusted devices. This is not user-friendly since the user has to process the same information twice. It also makes the user an unnecessary critical factor and risk in the security process. This paper concerns a case study of an information scheme which allows the user to enter critical information in a trusted device, which adds data necessary for the recipient to verify its integrity and authenticity. The output of the device is a code that contains the information and the additional verification data, which the user enters in the computer used for online banking. With this, the bank receives the information in a secure manner without requiring an additional check by the user, since the data is protected from the moment the user entered it in the trusted device. This proposal shows that mundane tasks for the user in online banking can be automated, which improves both security and usability.

1 INTRODUCTION

Security and usability are often seen as opposites. It is easy to sacrifice one in order to improve the other, but hard to improve one without affecting the other negatively. The work in this paper has the main goal to, in a very specific area, improve usability and (indirectly) security. This area is the secure creation of financial transactions by users in online banking.

Home computers are vulnerable to Man-in-the-Browser (MitB) attacks (Curran and Dougan, 2012). Smartphones and tablets are not exempt from malware attacks either (Felt et al., 2011) and are also vulnerable to MitB attacks if a browser is used to access a (mobile) bank site. User-owned are not trusted by banks since adversaries can use them to intervene in the communication flow between user and bank. Banks often rely on small devices given to their customers to have a trusted environment at the user's side (Poll and de Ruiter, 2013; Kiljan et al., 2014a). These devices are used for user authentication (using one-time passwords or challenge-response authentication) and for transaction verification.

The used information scheme in which users verify transaction data is known as What You See Is What You Sign (WYSIWYS), of which an overview is shown by Figure 1. WYSIWYS lets users verify transaction information in a trusted environment that a bank received previously from an untrusted environment. In step 1, the user enters transaction data in the computer used for online banking, which sends it to the bank using the Internet in step 2. Verification of whether the user and not an adversary provided the transaction information through the untrusted environment happens in step 3. The most important transaction information and a one-time password (OTP) are returned over a secure channel to the bank authentication device in possession of the user. The user either confirms or denies whether the data received in step 4 is the same as what was entered in step 1. A confirmation is given by repeating steps 1 and 2 with the OTP. The transaction information received in step

Kiljan, S., Vranken, H. and Eekelen, M.
User-friendly Manual Transfer of Authenticated Online Banking Transaction Data - A Case Study that Applies the What You Enter Is What You Sign Transaction Authorization Information Scheme.
DOI: 10.5220/0005965102590270
In *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016) - Volume 4: SECRYPT*, pages 259-270
ISBN: 978-989-758-196-0

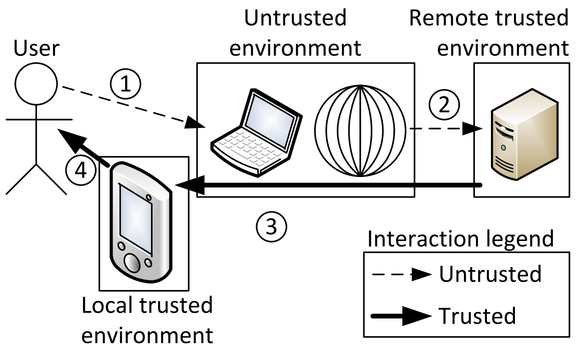

Figure 1: The information flow between environments with the What You See Is What You Sign authentication information scheme.

4 is what the user sees, and the user effectively signs the data when he or she sends the OTP back to the bank.[1] Note that only the user is capable of providing the OTP (effectively the signature) since it is not known by the untrusted environment. Returning the OTP through the untrusted environment is therefore not insecure. An aspect of WYSIWYS that is neither secure nor user-friendly is that the user is required to verify the same information twice: once upon entry (step 1) and once when it is returned (step 4). Users are quite unreliable in comparing numbers (AlZomai et al., 2008), and mistakes (genuine due to simply not seeing changed numbers, or abusive due to laziness) can be expected. We proposed an alternative information scheme named What You Enter Is What You Sign (WYEIWYS) (Kiljan et al., 2014b). With WYEIWYS, transaction data is entered immediately in a trusted environment before it is forwarded to the bank in a secure manner, which makes the second verification that WYSIWYS has unnecessary. Our original proposal misses an implementation which describes how the critical transaction data entered by the user is transferred from a trusted environment to an untrusted environment that will forward it to the bank. We suggested the use of a connection between the trusted authentication device and the untrusted computer used for online banking. Such a connection is troublesome. For security, it widens the attack surface of the trusted environment since it will be more exposed to the untrusted environment. In addition, online banking can be done with a plethora of different devices. The authentication device might not be capable of creating a connection to the computer used for online banking due to the lack of a compatible interface.

[1]This is unlike the use of the verb 'sign' in cryptography, where it refers to the use of asymmetrical encryption to sign a message with a private key, where the signature can be used to verify a messages authenticity, integrity and non-repudiation. Our definition of 'signature' for this paper is data which warrants the authenticity and integrity of other data, but not its non-repudiation.

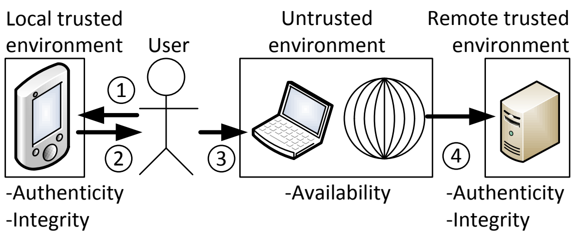

Figure 2: An overview of the path the transaction data follows (trusted from the very first step), and the information security principles for which each environment is responsible.

The challenge therefore is to apply WYEIWYS in a way that both increases usability and security (compared to WYSIWYS). We propose the use of a code to be transcribed by the user, which we refer to as a Message Code (MC). See Figure 2 for an overview. In step 1, the user starts with entering the critical transaction data in a trusted authentication device, which 'signs' it by adding data (the signature) that warrants authenticity and integrity. Both the data and the signature are converted to an MC. The MC is read (step 2) and transcribed into the untrusted environment (step 3), which forwards the data to a remote trusted environment (step 4). Integrity and authenticity of the data is protected as it passes through the untrusted environment.

The one-way information flow of reading a code from a display and writing it on another device physically and logically separates the local trusted environment from the untrusted environment, reducing the attack surface of the former. Since a connection between devices is not necessary, compatibility issues are non-existent as any untrusted user-owned device can be used.

Our contribution is an analysis and a case study of a process that allows users to transcribe a message and data which verifies the integrity and authenticity of the message together using a single code. In Section 2, the steps and methods that can be used to generate an MC in a user-friendly and secure manner are analyzed. The case study for an implementation of an MC to secure financial transaction online banking is described in Section 3. Furthermore, in Section 4 we reflect back on the case study, limitations of our work and possible further research. Section 5 closes with our concluding remarks.

2 ANALYSIS OF GENERATING AND VERIFYING AN MC

There are scenarios in which it is necessary for a service provider to know whether a specific user provides a certain piece of information, and that the information was not altered along the way (either accidentally or by an adversary). The amount of resources spend on this depends on the value of the information and how often the information flow occurs. For example, if an individual needs to register for a service that needs certainty about the user's identity, it is not enough to just assume that the named individual is actually the one he or she claims to be. If such a registration is only done once, it would be worthwhile to have the user visit a branch office of the service provider and provide identification documents, such as a passport. In this scenario, the branch office and the person assisting the user (both under control of the service provider) and the passport (provided by the government) ascertain the identity of the individual.

However, such physical interaction is unwanted if it concerns valuable information that the user sends more often to the service provider. An example is given by online banking. Users expect that they can make transactions anywhere at anytime. A trusted (from the perspective of the bank) environment is required that lets users provide information securely. There are two distinct trusted environments: the local trusted environment available to the user, and the infrastructure of the service provider. Between these environments is a large untrusted environment, where Man-in-the-Middle (MitM) attacks occur. These attacks do not only occur on the Internet (against which SSL/TLS can provide adequate protection), but also on users' computers. An example are Man-in-the-Browser (MitB) attacks, through which an adversary retrieves authentication credentials or silently changes information between user and service provider (Curran and Dougan, 2012). If users' devices cannot be trusted, the most obvious approach would be to provide each user a device that hosts a local trusted environment. Users should be able to use this device to provide required information in a secure manner.

Weigold and Hiltgen proposed several online banking transaction authentication methods (Weigold and Hiltgen, 2011). For one of their proposals, the user enters the same transaction details in an untrusted computer used for online banking and in a trusted device, provided by the bank and in possession of the user. The trusted device generates a one-time password (OTP), which the user enters on the untrusted computer used for online banking. The OTP from their proposal is created based on the information entered on the trusted device, and protects the information entered on the untrusted device. The information, entered twice by the user, should be equal to correspond with a valid TAC.

What we propose and analyze further in this section is the use of a Message Code (MC). An MC not only contains the information required to verify the integrity and authenticity of a message, but also the message itself.

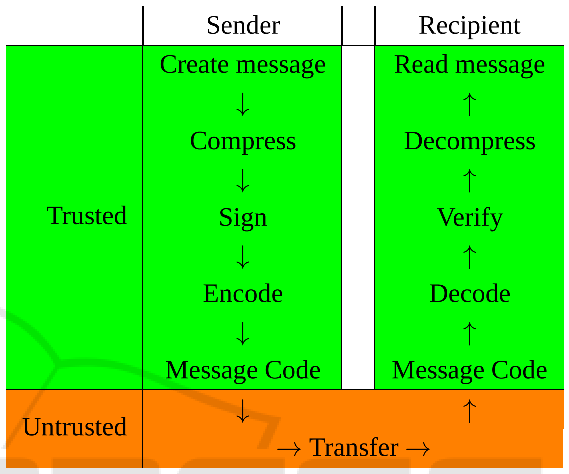

Figure 3: An overview of the process required to create a Message Code.

Figure 3 shows two trusted environments, and the actions required to send and receive a message securely using an MC. It is assumed that each party is the only one which can control its own trusted environment. The paired intermediate steps related to the processing of the message by both parties are explained in more detail.

2.1 (De)compression Algorithms

If applied correctly, compression reduces the size of messages. A message will be part of the MC. Reducing its size would imply a reduction in the MC's size, which is beneficial when the user has to transfer the information later.

Table 1 shows an overview of the effect that the Deflate[2] general-purpose compression algorithm has on random text of different sizes. For such small pieces of data a general-purpose compression algorithm can have either a positive or a negative effect, and is therefore unreliable. In the most optimal circumstance (in very specific cases of having 8 charac-

[2]RFC 1951 - DEFLATE Compressed Data Format Specification version 1.3: `https://tools.ietf.org/html/rfc1951`

ters of data), Deflate compresses data by 25%. Depending on the number of characters, the output can actually have a size increase between 6.3% (30 characters) and 28.6% (5 characters). This can be explained by that general-purpose compression algorithms build a dictionary of data to compress based on earlier compressed data. Therefore, small pieces of data do not compress really well (or in some cases, at all) with such algorithms.

An algorithm that specializes more on the compression of very short pieces of text could be used instead, and can be created by using a static dictionary of commonly used text parts.[3] However, the amount of compression it provides can be unreliable, depending on how well the message 'fits' the pre-established dictionary.

A completely custom algorithm can be designed instead, tailored to removing and reconstructing redundant information from messages that comply to a pre-established structure. For the algorithm to be durable, the messages' structure should not change.

2.2 Securing and Verifying Authenticity And Integrity

The recipient of an MC needs to be able to establish the authenticity of the message's source, and to verify that the message was not changed after it was sent. The sender has to prepare the data in such a way that the recipient will be able to perform the necessary checks. There are several ways in which the sender can make these preparations, and for the receiver to verify the message.

Digital signatures are an option. Using public key cryptography, the integrity and authenticity of a message can be established by adding a signature to the message. In addition, digital signatures provide non-repudiation: the sender cannot claim that he or she did not send a message that the recipient has received and validated. However, a drawback of digital signatures is that key management is quite complex.

Message authentication codes (MAC) are another option. A MAC is data added to a message with which inferences can be made about its integrity and authenticity, similar to a digital signature. Creating a MAC requires the data of which the integrity and authenticity should be protected (the message), and some secret data which provides the authenticity (a secret key). Verifying a MAC requires the message to be verified and a secret key known to the sender and the recipient. A difference with digital signatures is that a single key

Table 1: Output sizes of the Deflate compression algorithm, based on 10,000,000 randomly generated text values with a character range of [A-Za-z0-9], encoded in 8 bit ASCII.

Number of characters	Size (bits)	Compressed size (bits)
5	40	40-56
6	48	40-64
7	56	48-72
8	64	48-80
9	72	64-88
10	80	72-96
11	88	72-104
12	96	80-112
13	104	96-120
14	112	104-128
15	120	104-136
16	128	112-144
17	136	112-152
18	144	128-160
19	152	136-168
20	160	136-176
21	168	144-184
22	176	160-192
23	184	160-200
24	192	176-208
25	200	184-216
26	208	192-224
27	216	200-232
28	224	200-240
29	232	216-248
30	240	224-256

is used. This simplifies key management somewhat since all involved parties (sender and recipient(s)) use the same key, but it sacrifices non-repudiation since it cannot be proven which party signed a message.

With MACs, it is initially required that the trusted environment that generates the key sends it to the other trusted environment, without an intermediate untrusted environment. Otherwise a man-in-the-middle could intercept the key. A public key infrastructure with trusted third-parties that issue certificates to create digital signatures might be a better choice if it is not viable to securely transfer a secret key from one trusted environment to the other at the beginning of the authentication device's lifespan.

Note that confidentiality is not a security principle used in this analysis, nor in the case study in Section 3. If it would be, authenticated encryption would be one approach which combines confidentiality, integrity and authenticity.

Whether a digital signature or MAC is used, it is important to provide protection against replay attacks. A cryptographic nonce (number used once) can provide protection against such attacks if it is used to generate the digital signature or MAC. Note that a nonce does not have to add data to the message itself as long as the remote trusted environment is able to reconstruct the nonce when verifying the message. Examples of such nonces include time stamps and counters.

2.3 Encoding/Decoding Methods

Compressed data, digital signatures and MACs often do not consist exclusively of human readable data. An encoding can convert data to human readable and writable text. There are several approaches. Wiseman et al. performed a comparison of three distinct encoding schemes for one-time passwords used in device pairing (Wiseman et al., 2016). These will be referred to as Wiseman's word encoding, Wiseman's alphanumerical encoding and Wiseman's numerical en-

[3]An example is SMAZ, compression for very small strings: `https://github.com/antirez/smaz`

coding. Each encoding was tested on its efficiency and perceived usability when transcribed by users on home computers and mobile devices. A lower limit was set on the length of the codes of 500 million possible combinations for each tested encoding scheme. In addition, codes were padded when required to always give a fixed length.

For Wiseman's word encoding, the fixed length is 3 words of 3 letters. An index of 800 words was used. Each of the three words in a code represents $log_2(800) \approx 9.644$ bits.

For Wiseman's alphanumerical encoding the fixed length is 5 alphanumeric characters using a set of 56 different characters from the range [a-zA-Z0-9], excluding 'i', 'o', '1', 'I', 'O' and 'L'. Each character in the code represents $log_2(56) \approx 5.807$ bits. There are also other alphanumerical encoding schemes. Base64 is probably the most famous encoding scheme that outputs 'real' characters exclusively (that is: characters that most humans can see and interpret, assuming that they are familiar with the English alphabet), and each of its 64 characters represents a 6 bit value.[4] Base32 is a variation which uses 32 characters instead and avoids the use of special and mixed case characters by only using upper case letters and some digits. Each of Base32's characters represents 5 bits. The same binary data encoded in Base32 will be represented by more characters compared to Base64. A variation of Base32 is Z-Base 32.[5] The most differentiating aspect of Z-Base 32 is its use of lower case characters.

Finally, there is Wiseman's numerical encoding, which presents all data as a base 10 number with a fixed length of 9 digits. Each digit represents $log_2(10) \approx 3.322$ bits.

Wiseman et al. concluded that words are the easiest to transcribe. It must be noted that they were only interested in using an encoding scheme to create a one-time password. This can be compared to the security information required to validate the integrity and authenticity of data, as discussed in the previous section. However, an MC will contain both the data itself and this security information, implying that its length will be longer compared to that of a one-time password in any encoding scheme. This is something that should be kept in mind when making a decision on what kind of encoding should be used.

The output of the encoding phase is an MC. Its characters can be grouped to make them easier to transcribe. For example, words can be separated by spaces and characters of alphanumerical codes can be separated by dashes.

[4]RFC 4648 - The Base16, Base32 and Base64 Data Encodings: `https://tools.ietf.org/html/rfc4648`

[5]Human oriented base-32 encoding - O'Whielacronx (2009): `http://philzimmermann.com/docs/human-oriented-base-32-encoding.txt`

2.4 Code Transfer

After an MC is made, it needs to be transferred from the trusted environment. The MC is meant to reach another (remote) trusted environment. Different pathways can be taken, and most will use an untrusted environment (as shown in Figure 3). If that is the case, it is expected that the untrusted environment forwards the code to another trusted environment. The untrusted environment provides availability while the MC provides integrity and authenticity.

The MC can be read by the user from one (trusted) device and entered in another (possibly untrusted). To make this process user-friendly the MC should be as short as possible and consist of characters that are easy to read and write. The compression phase mostly dealt with the MC's length, while the encoding phase focused on the text that is actually to be transferred by the user.

Typographical mistakes can be detected by the recipient due to that the code contains both the message and additional data to verify its integrity and authenticity. When a mistake is made, the message will not correspond with the additional data when it is verified. The recipient refuses to process the message further and notifies the user to correct his entry. This further improves user-friendliness, since the user can make and correct mistakes without repercussions.

This paper focuses mostly on a human-transferable MC since it is the most universal method to transfer information from a trusted device to any untrusted device that allows user input, independent from used software that the latter runs. However, user-friendliness can be improved without sacrificing security in specific scenarios, depending on the used untrusted device. Most smartphones have a rear-facing camera which can be used to scan QR codes. An MC could be converted to a QR code that can be scanned by an application that forwards the data to the trusted environment. A QR code would aid in cases where user input is less user-friendly, such as with touch keyboards on smartphones (Gallagher and Byrne, 2015).

3 ONLINE BANKING CASE STUDY

This section notes a case study for the feasibility of using a Message Code (MC) for transaction authentication in online banking using the methods described in Section 2. It first explains why the use of an MC can be beneficial for security and usability in online banking before describing each step of its generation and verification.

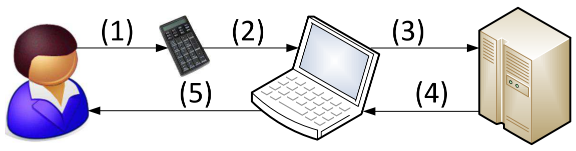

Figure 4: An overview of the What You Enter Is What You Sign information scheme.

The user's time should be used sparingly for security actions (Herley, 2009), so it might be beneficial if the user would not have to process transaction details twice. We proposed WYEIWYS (Kiljan et al., 2014b) as shown in Figure 4. The user enters critical transaction data (the destination account number and the amount) in a trusted device in step 1. The trusted device adds a digital signature and enters all data in the user's computer in step 2. The user's computer forwards the data to the bank in step 3. After verifying the signature, the bank reports the validity of the entered values back to the user in steps 4 and 5. The idea behind WYEIWYS is that a man-in-the-middle will not be able to change the data between steps 2 and 3 without the bank noticing it, since the digital signature ensures the integrity and authenticity of the entered critical transaction data. As shown in Figure 4, the secure information flow is one way only and the user is not expected to perform any checks afterwards.

The proposal does not specify which technology should be used in step 2. Keyboard emulation was suggested as one possible method, but this does require a connection and assumes that the user's computer (the untrusted device) actually supports keyboards. Connecting devices (wired or wirelessly) can also be quite cumbersome.

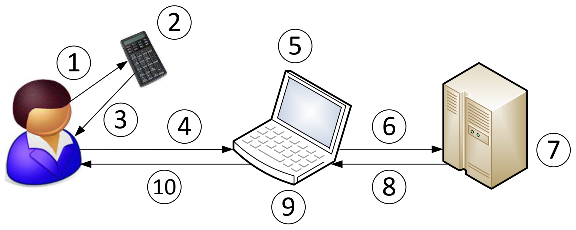

Figure 5: Overview of the information flow of WYEIWYS using a Message Code.

An MC could be used for transferring critical transaction data in a way that reduces compatibility issues. Figure 5 shows the information flow if the user would facilitate the transfer of the MC between both devices. In step 1, the user enters critical transaction information in the trusted device, which creates an MC in step 2 that is shown to the user in step 3. The user enters the MC into his home computer or mobile device in step 4. A browser or application receives the MC in step 5 and forwards it to the bank in step 6. The bank processes the MC in step 7 and returns the resulting data back in step 8 if the MC is valid. The data is processed by the user's computer in step 9 and shown to the user in step 10. If the MC is invalid, an error would be returned through steps 8, 9, and 10, after which the user can correct any typographical mistakes made in step 4.

This section continues with the considerations of constructing an MC in step 2 and the deconstruction in step 7. Looking at Figure 3, the authentication device in possession of the user would be the trusted environment available to the sender, the trusted environment of the recipient would be the bank's infrastructure, and the untrusted environment between them would be represented by the user's computer and the Internet. The authentication device is used to create the message and compress, sign and encode it to create an MC. The bank decodes, verifies the signature and decompresses the data.

3.1 Create the Message

We consider the destination account number and amount of money to be critical transaction information. Both are entered by the user in step 1 in the authentication device.

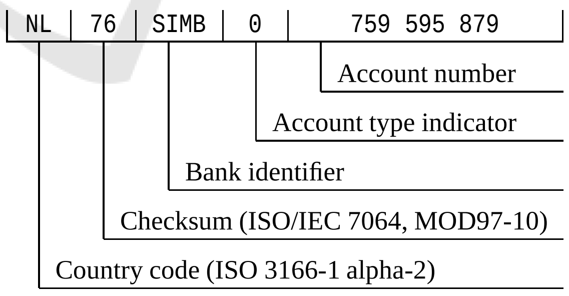

Figure 6: The structure of a Dutch IBAN account. The example IBAN account number is at the fictional **SIM**ulation **B**ank and has a valid checksum.

For the example, we assume that a Dutch IBAN is used as a destination account number. The structure of a Dutch IBAN is shown with an example in Figure 6. The country code and checksum always have the same location and length for all IBANs. The checksum also always has the same location and length. For Dutch account numbers, the bank identifier consists of four uppercase letters, the account type indicator is 0 for payment accounts and the account number is always nine digits.

The amount of money is a value with two decimal digits. For our proposal, we assume that a transaction has a maximum value of less than one million euro (so a maximum of € 999,999.99). We also assume that this method will only be used for domestic transactions (from a Dutch account to another Dutch account), so the user does not have to specify an alternative currency. An example value the user could enter is:

```
123456.78
```

To enter all values fully, the authentication device would require a keyboard which allows the user to enter [A-Z0-9] and possibly a decimal separator. Typographical mistakes in the IBAN can be detected by the device by verifying the checksum. Such a check is not possible on the amount of money, but the user can notice typographical mistakes when shown on the screen of the authentication device, and later (after step 10) on the display of his or her computer.

3.2 Create the Message Code

For step 2, we define several phases to generate an MC from the data entered in step 1.

- A compression phase, to compress user provided data. This shortens the MC in the end.
- A signature phase, to generate data to protect the message's authenticity and integrity.
- An encoding phase, which creates a string value from the data which a user can process.

3.2.1 Compress

There are several approaches for compression noted in Section 2.1. Data in online banking transactions is quite structured and well defined. Therefore, an example is given for a custom algorithm, tailored towards the data to be transferred. The functions described in Section 2.1 will be used: removal and reconstruction of redundant information, and restructuring data types for more efficient transportation.

To start, we use the IBAN from the example in Figure 6 and an amount of € 123456.78:

```
NL76SIMB0759595879  12345678
```

We removed the decimal separator from the amount to make it an integer value, which reduces complexity for further processing.

We already stated that we will only use this method for domestic transfers, so the country code can be removed. The checksum in the next two digits of the IBAN is only meant to protect against typographical mistakes. It has already been verified during entry in step 1, which is why we can also remove it. As discussed earlier, a Dutch IBAN has a bank identifier consisting of four upper case letters. Banks in the Netherlands can reconstruct the destination bank based on solely the account number, which is why we can also remove it. Finally, the leading 0 in front of the bank account number specifies the type of account. Any other number would be a savings account. Savings accounts do not support direct withdrawals or deposits from payment accounts owned by other account holders, which is why we can also remove this digit. This leaves us with:

~~NL~~ ~~76~~ ~~SIMB~~ ~~0~~ 759595879 12345678

The data in strikethrough text has been removed, and needs to be reconstructed in step 5.

Dutch account numbers have a maximum value of 999,999,999, which can be represented as an unsigned integer by 30 bits.[6] The amount (as an integer) has a maximum of 99,999,999, and can be represented as an unsigned integer by 27 bits. In binary, account number *an* and amount *am* are as follows (most significant bit first):

$$an = \texttt{101101010001101000001101100111}$$
$$am = \texttt{000101111000110000101001110}$$

Since the length of the bit string of each value is fixed, the bit strings can be concatenated to create a single string that can be split again in step 7 by the bank. This concatenated string is the raw message which must be processed further, and is 57 bits long.

3.2.2 Sign

For our example, there are only two parties involved who need to access key material: the authentication device (to create data for verifying the integrity and authenticity of the message) and the bank (to perform the verification). Based on the various methods noted in Section 2.2, a fitting approach would be the use of a MAC. It can be assumed that the bank can embed a shared secret key in the authentication device in a secure environment before it is given to a customer, and there are no other parties involved which would warrant the use of a public key infrastructure.

To prevent replay attacks, the MAC should be based on both the message and a nonce. The nonce itself is not secret, but should be unique and fresh for

[6]Actually, the maximum value would be 999999990 due to the inclusion of a checksum known as the 'elfproef', but for the sake of simplicity for this paper, we assume the maximum is the maximum value provided by 9 digits.

each MC. We assume a nonce is used that does not have to be part of the message (to keep it as short as possible). This could be a nonce based on synchronized clocks between authentication device and the bank, or an incrementing counter of which the bank keeps track.

To let the authentication device calculate MAC mac_a over message m (*an* and *am* concatenated, from the previous step), we use a randomly generated key k of 2048 bit, a 32-bit unsigned incrementing counter as nonce n_a, and a hash function H. Hence, we could use:

$$mac_a = H(\ k \mid n_a \mid m\ )$$

Care must be taken with this approach, since it is susceptible to length extension attacks if H is based on the Merkle-Dåmgard construction (such as MD-5, SHA-1 and SHA-2) (Sobti and Geetha, 2012). A mitigation would be to use an alternative hash algorithm. SHA-3 is based on the sponge construction, which does not have this inherit limitation.[7] Therefore, SHA-3 will be used for H for this example.

The strength of the MAC is based on its length. A larger MAC is more secure against brute-force attacks. However, since the MAC is part of the MC, a longer MAC would result in a longer MC that needs to be transcribed by the user. The length further depends on the chosen encoding of the MC, since a MAC with an aligned length prevents padding and therefore makes optimal use of the value of every character. The chosen encoding is Z-Base 32, which represents 5 bits for each encoded character. Z-Base 32 is further discussed in Section 3.2.3.

The message is 57 bits, as discussed in the previous section. A 23 bit MAC would make the total length 80 bits, which aligns with Z-Base 32 to be 16 characters. There is a 50% probability that a valid MAC is guessed if an adversary guesses 2^{22} times. The number of guesses can be limited by the bank, which can count the number of failed guesses and delay or block further attempts. This is discussed more in detail in Section 3.6.3.

The values and the output of the formula are:

[7]The Keccak sponge function family: `http://keccak.noekeon.org/`

$$\begin{aligned} k &= \texttt{0x00 0x01...0xFF} \\ n_a &= \texttt{0x00 0x00 0x00 0x01} \\ m &= \texttt{0x01 0x6A 0x34 0x1B 0x38 0xBC 0x61 0x4E} \\ mac_a &= H\{\ k \mid n_a \mid m\ \} = \texttt{0x77 0x64 0xCE} \end{aligned}$$

3.2.3 Encode

The compression phase was used to compress the data and turn it into a bit stream. Furthermore, we gave an example of how a MAC can be calculated in the signature phase and computed a MAC of 23 bits. In the encoding phase, we combine the values from both phases to create an MC, a human-readable text string that the authentication device will show on its display.

The output of the compression phase is 80 bits. Based on the encoding methods noted in Section 2.3 it is possible to tell how long the MC will be by calculating how much words or characters would be needed to encode it, rounding up to ensure a fixed length. If the 800 word encoding from Wiseman et al. would be used (Wiseman et al., 2016), the length would be 9 words or 27 characters without separators. Base64 would require 14 characters, and Base32 and Z-Base 32 would require 16 characters, all excluding separators. 25 characters would be required if only digits were used.

To improve readability, separators would be required to create groups of characters. For words, that would be a total of 35 characters (based on 8 spaces). Based on groups of 4 characters each, three separators would have to be added to Base64, Base32 and Z-Base 32, bringing their number of characters respectively to 17, 19 and 19. When only digits would be used and 4 separators would be used to create groups of 5 digits, 30 characters would be required. These are the number of characters that the authentication device's display would be required to show.

Wiseman et al. (2016) tested their word encoding with 3 words, which was the preferred encoding by their test candidates. However, 9 words would be quite long to show on an authentication device and for users to enter on other devices. For this case study we will use Z-Base 32 instead, which provides a good balance between character length and recognition of the used characters.

See Table 2 for an overview of the encoding pro-

Table 2: Creating a single human readable string from the message and the MAC using Z-Base 32.

	Message											MAC				
	Destinat. account: 759595879 (30 bits)						Amount: 12345678 (27 bits)					0x7764ce (23 bits)				
Bits	10110	10100	01101	00000	11011	00111	00010	11110	00110	00010	10011	10111	01110	11001	00110	01110
Decimal	22	20	13	0	27	7	2	30	6	2	19	23	14	25	6	14
Z-Base 32	s	w	p	y	5	8	n	6	g	n	u	z	q	3	g	q

cess using Z-Base 32. The result is an MC, which could be displayed as:

`swpy-58n6-gnuz-q3gq`

3.3 Transfer the Message Code

As described in Section 2.4 and as shown by steps 3 and 4 in Figure 5, the user reads the MC from the trusted authentication device and enters it in the untrusted computer used for online banking. To make the transfer user-friendly, the MC's length was reduced by compressing the message in Section 3.2.1 while it is structured by the alphanumerical encoding chosen in Section 3.2.3.

If a mobile banking application on a smartphone or tablet is used instead of a home computer, an alternative to the manual transfer by the user of the MC from the authentication device to the mobile device might be the use of a QR code if the device has a rear-facing camera. The authentication device would show the QR code to be scanned and the mobile banking application would scan the QR code. An advantage of this would be that the user is not required to manually enter the MC as a text string into the mobile device. Compression can still be applied to keep the QR code as small as possible.

3.4 Verify the Message Code

Step 7 of Figure 5 concerns the verification of the MC, and is mostly step 2 in reverse order. First, the data is decoded back into a bit stream and the message and MAC are separated. After that, the bank needs to verify if the included MAC corresponds with the payload to verify integrity and authenticity. Only if both are verified will the message be processed further.

3.4.1 Decode

Decoding is the reverse of encoding as done at the end of Step 2. Table 2 can be read starting from the bottom row to get an idea of the decoding process. The result is the top row's bitstream.

3.4.2 Verify

The bank calculates its own MAC, and with that effectively performs the same steps as the authentication device did in step 2. The assumption is that the bank has access to the same k and is able to deduce n_a. A counter was used for n_a, which increases by one for every generated MAC. The bank has to store the nonce of the last received valid message to detect replay attacks in future messages, which we will refer to as n_b.

n_a as a 32-bit value is not included in the message to keep the MC as short as possible. That is why the bank has to deduce it. It is possible that the user generates MACs which the bank never receives. This can happen if the user is testing the workings of the authentication device or when an online banking session is disconnected, after generating the MAC but before the bank receives it. Since n_a is a counter that is only incremented by the authentication device, all the bank has to do to compensate for discrepancies between the last stored value in the authentication device and at the bank would be to attempt to verify the MAC multiple times with increasing values for the nonce, starting at n_b and increasing for an acceptable number of messages which the bank did possibly not receive. It might be likely that the user created a message or two that were missed, but it is unlikely that the user generated 50 messages. In this exceptional scenario, the user could be requested to contact the bank.

For our example, we assume that no previous MCs were missed by the bank. Let k be the shared key between authentication device and bank, n_b the earlier mentioned stored nonce value at the bank (increased to the value of $n_a + 1$ with each valid received message), m the message, and mac_b the MAC that the bank calculates.

$$\begin{aligned} k &= \texttt{0x00 0x01...0xFF} \\ n_b &= \texttt{0x00 0x00 0x00 0x01} \\ m &= \texttt{0x01 0x6A 0x34 0x1B 0x38 0xBC 0x61 0x4E} \\ mac_b &= H\{\, k \mid n_b \mid m \,\} = \texttt{0x77 0x64 0xCE} \end{aligned}$$

Now all the bank has to do is verify if mac_a equals mac_b. If they are equal, the message is authentic and its integrity is protected, and should therefore be processed further.

3.4.3 Decompress

Decompression of m is the opposite operation of compression as done in step 2. First, the two values are separated, based on their lengths and positions. Let an again be the destination account number and am the amount of money.

$$\begin{aligned} an &= \texttt{101101010001101000001101100111} \\ am &= \texttt{0001011110001100001010011110} \end{aligned}$$

an is converted to an integer which results in a value of 759,595,879. The IBAN is reconstructed by supplementing the integer with known values. In the end, the bank wants to have the same number as shown in Figure 6.

The country code is easy since the transfers were limited to Dutch domestic accounts. Therefore, the bank knows the country code is `NL`. As noted earlier, banks in the Netherlands can identify which account number belongs to which bank, allowing them to match the account number to the bank with the bank code `SIMB`. Finally, the account type indicator is always `0` for payment accounts. The value the bank now has is:

```
NL ?? SIMB 0 759 595 879
```

The IBAN checksum is still missing, which can be recalculated.[8] The fully reconstructed IBAN:

```
NL 76 SIMB 0 759 595 879
```

All that is needed to reconstruct the amount *am* is a conversion to an unsigned 32-bit integer and a division by 100 to create the original decimal value. With domestic transfers it is not required for the user to specify the currency if there is only one. Our example concerns the Netherlands, which uses the euro. Therefore, the bank knows that the value to transfer is € 123456.78.

3.5 Further Processing

The bank returns the IBAN and the amount back to the user's browser (step 8 of Figure 5), which receives (step 9) and shows (step 10) them. The user has the opportunity to fill in the rest of the non-critical transaction values and submit the transaction. An overview screen could be shown before the user gives his final approval. Note that this final approval would concern usability (the user has one more chance to correct any mistakes), and unlike WYSIWYS there is not a mandatory secondary check of initially entered data as a security action that the user has to perform. If a user already checked the information on correctness when it was entered, it would not be required to thoroughly check an overview of the data before final approval is given.

3.6 Attack Analysis

The security of the system is based upon the generation and verification of the MAC, which provides integrity ('Was the data changed in any way?') and authenticity ('Does the data come from the expected source?'). An adversary not having access to the key should be unable to generate valid MACs at will.

[8] European Committee for Banking Standards (August 2013) - IBAN: `http://www.europeanpaymentscouncil.eu/documents/ECBS IBAN standard EBS204_V3.2.pdf`

3.6.1 The Random Adversary

Steps 5 and 9 of Figure 5 are where malware could modify transaction data before it is sent to the bank and the bank's reply before it is returned to the user. Imagine that an adversary wants to change the destination account number silently at the beginning of step 5. An account controlled by the adversary is NL 38 VIRB 0 307 633 357 (at the fictional **VIR**tual **B**ank). The attacker modifies the data as follows (presented in their most clear data types for clarity) in step 5:

	Account	Amount	MAC
Original:	`759595879`	123456.78	0x77 0x64 0xCE
Modified:	`307633357`	123456.78	0x77 0x64 0xCE

When only the account number is changed in step 5, the bank would notice upon processing the data in step 7 that the generated MAC is not based on the received account and amount data. If the MAC does not fit the message (account and amount), the transaction is discarded. For steps 8 and onward, the bank might notify the user to contact the bank for clarification.

3.6.2 The Known Adversary

A 'known' adversary is an adversary to which the user (for a legitimate reason) transfers money to, either at the moment of an attack or in the past. The known adversary, having full control of the user's computer, can attempt to use an older valid transaction code of a transaction to the adversary to create a new transaction in a replay attack.

Due to the inclusion of a nonce in the MAC, it is possible for the bank to detect a replay attack. In our example, we used a counter that increases by one for each generated MAC. A bank could detect a replay attack by storing n_b, the nonce of the last received legit message. Attempts to verify the MAC of the replayed message would fail since n_a would be lower than n_b, and the bank only checks the current and higher n_b values.

Another potential attack vector exists with a known adversary. If the user prepares a legitimate transaction to the adversary, the adversary could change the confirmation that is given in step 10 to the user. Through control of the user's computer, the adversary could change the verification information in step 9 and make the user think that the transaction failed, which could make the user re-authenticate the transaction and send it again. The re-attempt would concern a second (illegitimate) transaction. Banks could provide protection against this by monitoring repeated transactions to the same destination account number within a certain time frame.

3.6.3 The Determined Adversary

The used MAC is relatively short (23 bits in the example), to keep the MC as short as possible and to make it align with the 5 bit boundary of Z-Base 32. A determined adversary might attempt to brute force a valid MAC, since for every payload it can be expected that one valid MAC exists in the range of 2^{23} and since the adversary has a 50% probability of guessing a correct MAC when 2^{22} attempts are made.

What limits an adversary is that the MAC can only be verified by the bank (due to missing the secret key required for generating and verifying the MAC). The bank can register accounts for which repeatedly wrong MACs are generated, and not process further attempts from them for a specified amount of time. An example could be a policy that limits the number of wrong MACs to ten, after which further attempts are hindered (such as, the customer of the account has to contact the bank). With ten attempts, an adversary has a $100\% * \frac{10}{2^{23}} = 0.00012\%$ chance of success.

Brute forcing a transaction to a random destination address would not aid the adversary. In the unlikely event that a valid MAC would be generated for one message, it would not disclose the key required to generate MACs for another message with a different account number. Therefore, the destination address provided by an adversary would most likely be under control of the adversary. Further transactions to a destination address for which multiple wrong MACs are generated can be delayed or blocked by the bank.

4 DISCUSSION, LIMITATIONS AND FURTHER RESEARCH

In Section 2 we analyzed methods that can be used to create an MC. The case study in Section 3 shows that it is possible to use a selection of these methods to create and verify an MC for transaction authentication in online banking in a way that is device-independent and does not rely on the user to make redundant security decisions. Note that this is a case study which only addresses the challenges faced in a single scenario. Other scenarios could present different obstacles and different methods might be more appropriate to overcome these using an MC, or the use of an MC might not be appropriate at all. For example, the case study applied the WYEIWYS information scheme, which is only usable if the user provides transaction information and not a third-party (such as an online e-commerce system). In the case of the latter, WYSIWYS might still be the best method to authorize such transactions.

Further research could experiment by testing the (perceived) usability when transferring MCs from one device to another using different manual methods (alphanumerical, numerical, words-based), and possibly expand on the idea of using automated methods whenever the use case allows it (such as QR codes through mobile devices). Another point for further research is mitigation of the attack vector described at the end of Section 3.6.2, in which an adversary can trick a user to perform a legitimate transaction again as a fraudulent transaction. An example of such a mitigation might be the authentication of a bank's reply. In addition, further research could examine whether WYEIWYS using a code would be usable in other scenarios in which a user needs to supply critical information using an untrusted environment, such as for providing a phone number for (further) out-of-band authentication.

5 CONCLUDING REMARKS

We proposed a method that allows humans to transfer both a message and data to secure the integrity and authenticity of the message by transcribing a single code. Different methods were examined to construct such a code in a way that makes the transfer user-friendly. In addition, a case study was performed in which such a code was used to secure online banking transactions for which the user provides critical information.

The case study shows that for online banking, the use of a Message Code can remove the necessity for a user to verify entered information twice, as is currently done. By taking away a critical decision from the user, usability is improved since the user has less critical choices to make, which also improves security since authenticity and integrity of the data rely less on user activities.

ACKNOWLEDGMENTS

This article is a product of the Dutch Research Program on Safety and Security of Online Banking. The research program is funded by the Dutch banking sector (represented by the Dutch Banking Association), the Police Academy, and the Dutch National Police.

REFERENCES

AlZomai, M., AlFayyadh, B., Jøsang, A., and McCullagh, A. (2008). An exprimental investigation of the usability of transaction authorization in online bank security systems. In *Proceedings of the sixth Australasian conference on Information security-Volume 81*, pages 65–73. Australian Computer Society, Inc.

Curran, K. and Dougan, T. (2012). Man in the Browser Attacks. *Int. J. Ambient Comput. Intell.*, 4(1):29–39.

Felt, A. P., Finifter, M., Chin, E., Hanna, S., and Wagner, D. (2011). A Survey of Mobile Malware in the Wild. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, pages 3–14, New York, NY, USA. ACM.

Gallagher, M. A. and Byrne, M. D. (2015). Modeling Password Entry on a Mobile Device. In *Proceedings of the International Conference on Cognitive Modeling*.

Herley, C. (2009). So Long, and No Thanks for the Externalities: The Rational Rejection of Security Advice by Users. In *Proceedings of the 2009 Workshop on New Security Paradigms Workshop*, NSPW '09, pages 133–144, New York, NY, USA. ACM.

Kiljan, S., Simoens, K., De Cock, D., van Eekelen, M., and Vranken, H. (2014a). Security of Online Banking Systems. Technical Report TR-OU-INF-2014-01 (Open Universiteit).

Kiljan, S., Vranken, H., and Van Eekelen, M. (2014b). What You Enter Is What You Sign: Input Integrity in an Online Banking Environment. In *Socio-Technical Aspects in Security and Trust (STAST), 2014 Workshop on*, pages 40–47.

Poll, E. and de Ruiter, J. (2013). The Radboud Reader: A Minimal Trusted Smartcard Reader for Securing Online Transactions. In *Policies and Research in Identity Management - Third IFIP WG 11.6 Working Conference, IDMAN 2013, London, UK, April 8-9, 2013. Proceedings*, pages 107–120.

Sobti, R. and Geetha, G. (2012). Cryptographic Hash Functions: A Review. *International Journal of Computer Science Issues*, 9(2):461–479.

Weigold, T. and Hiltgen, A. (2011). Secure confirmation of sensitive transaction data in modern Internet banking services. In *Internet Security (WorldCIS), 2011 World Congress on*, pages 125–132.

Wiseman, S., Mino, G. S., Cox, A. L., Gould, S. J., Moore, J., and Needham, C. (2016). Use Your Words: Designing One-time Pairing Codes to Improve User Experience (to be published). In *Proceedings of the 34rd Annual ACM Conference on Human Factors in Computing Systems*. ACM Publications.