Underwater Confidential Communications in JANUS

Yannick Beaupré[1(✉)], Michel Barbeau[1], and Stéphane Blouin[2]

[1] School of Computer Science, Carleton University, Ottawa, Canada
yannick-beaupre@outlook.com,
barbeau@scs.carleton.ca
[2] Defence R&D Canada Atlantic Research Centre, Dartmouth, Canada
Stephane.Blouin2@forces.gc.ca
https://carleton.ca/scs/

Abstract. JANUS is a NATO underwater communications standard. In this paper, confidential communications for JANUS are explored. The focus is on work published in 2021, Venilia and Tiny Underwater Block cipher (TUBcipher). Venilia consists of protocol elements that had been developed to provide confidential JANUS underwater communications. It leverages TUBcipher, a symmetric cryptography scheme. Some important details were not considered in this original work, including the security analysis of TUBcipher and key establishment and distribution. Intending to improve Venilia, these two points are addressed in the current paper. A security analysis is conducted for TUBcipher. TUBcipher has been implemented. Entropy data of ciphertexts have been collected with the ENT tool. Our results show that TUBcipher achieves close to the theoretical maximum entropy value. Two key establishment/distribution solutions are introduced. They provide methods of practically implementing Venilia within underwater communication networks.

Keywords: Underwater communications · confidential communications · security protocol · JANUS · Venilia · TUBcipher

1 Introduction

In contrast to classical networks, underwater communication networks present unique challenges. They involve sending messages through multitudes of nodes within large bodies of water. It is a harsh environment for communications [17]. Electromagnetic waves do not propagate well through bodies of water [4]. Acoustic waves are used to send messages. While they propagate in water, bandwidth is narrow, and packets that can be sent are limited in size. Furthermore, acoustic waves travel at a much slower speed than electromagnetic waves. It is an environment where packet processing performance is not critical due to the relatively low achievable data rates, particularly for long distances. However, the optimal use of a small packet size is a critical aspect.

G.-V. Jourdan et al. (Eds.): FPS 2022, LNCS 13877, pp. 255–270, 2023.
https://doi.org/10.1007/978-3-031-30122-3_16

JANUS, the Roman God of openings and gateways, is the name of an underwater communication standard developed and tested by the North Atlantic Treaty Organization (NATO)'s members. It specifies a means of encoding data within acoustic waves [15]. The packet size is 64 bits. Such a small packet size is not adapted to modern cryptographic methods that require a relatively large block size. For example, the Advanced Encryption Standard (AES) [12] requires a minimum block size of 128 bits. Thus, a small block cipher, called TUBcipher, has been created to work alongside a new JANUS data block class called Venilia [8].

The Venilia protocol is one of a kind. It aims to achieve confidential underwater communications. However, there are important aspects of it that deserve further investigation, such as the security analysis of TUBcipher. In this paper, the security of Venilia is analyzed. Concrete measurements of security metrics on TUBcipher are presented. They enable making statements about the security provided by TUBcipher, in comparison to other small block ciphers. It is concluded that TUBcipher does not achieve perfect indistinguisability, but does reach near maximum entropy. Ideas to improve the current Venilia solution are developed. A problem complementary to confidentiality is dynamic key distribution. Two key distribution protocols to be used with Venilia are created, a simple approach and a more complex one.

Related work is reviewed in Sect. 2. Section 3 describes the security analysis conducted for TUBcipher. The two key distribution solutions are discussed in Sect. 4. Section 5 concludes.

2 Background Work

JANUS, the Venilia class, and TUBcipher are briefly introduced.

2.1 JANUS

JANUS is a NATO communication protocol for underwater assets [15]. It aims at manufacturer independence and interoperability within networks containing assets of different suppliers. It was developed to become an underwater communications standard.

JANUS packets are smaller than typical network packets because of the harsh underwater propagation environment. The tiny sizes of packets make traditional cryptographic schemes unfeasible for use with JANUS. For reference, a typical UDP or TCP packet has a size of 1,500 bytes, that is, 12,000 bits [6]. Thus, the 128-bit block size of AES, a popular and strong cryptographic scheme, does not cause any problems for typical internet communications. However, JANUS packets have a size of 64 bits. Every packer comprises a 34-bit data block, over three times smaller than the required block size of AES. Given this limitation, a user class named Venilia, was developed for confidential JANUS communications [8]. A user class designates the purpose of the communication. Different

user classes handle the 34-bit data blocks of packets differently. Venilia is proposed as user class id 17 for confidential JANUS communications. The Venilia class is described in more detail in the upcoming section.

2.2 Venilia and TUBcipher

A solution to JANUS' lack of security has been proposed as a new class named Venilia [8] and a small block cipher called TUBcipher [9].

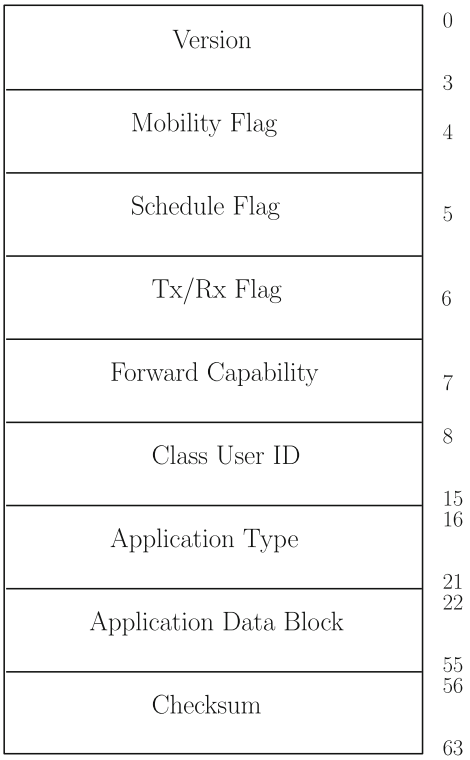

Fig. 1. 64-bit JANUS packet.

A JANUS packet has 64 bits, of which only 34 bits are utilized for a data block, while the rest is overhead, see Fig. 1. According to the suggested behavior of the Venilia class, there is an eight-bit Pre-Canned Message field, which contains an index for one of 256 unique messages predefined in the Venilia code book [8].

It is followed by the fields Destination ID (7 bits), Source ID (7 bits), and Inner Cyclic Redundancy Check (CRC) (5 bits), see Fig. 2.

Venilia adopted a code book approach to message content coding [8]. Given that only eight bits are used for the message in a packet, an integer from zero

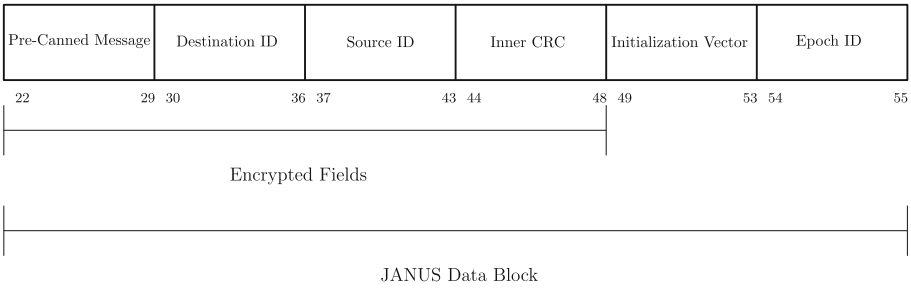

Fig. 2. JANUS class 17: Venilia data block.

to 255 is sent. This integer corresponds to a message for command and control or status communications.

Confidentiality is achieved through partial content encryption. Only the first 27 bits of the data block are encrypted using TUBcipher, i.e., the fields Pre-Canned Message, Destination ID, Source ID, and Inner CRC. Hence, encryption requires a block size of only 27 bits [9]. TUBcipher is a deterministic cryptosystem. The remaining seven bits of the 34-bit data block comprise a nonce made of an Initialization Vector (IV) (5 bits) and an Epoch ID (2 bits). The nonce is part of a mechanism for replay attack detection [19].

TUBcipher is a symmetric small block cipher based on PRINTcipher, originally developed for integrated circuit printing [11]. TUBcipher is a substitution-permutation network that encrypts a 27-bit block with a 2560-bit key over 56 rounds [9]. The 2560-bit key is generated using a concatenation of an epoch ID, a 256-bit key, an IV, and an eight-bit counter. This concatenation is hashed with SHA512. This process is repeated five times, every time incrementing the counter. All five hashes are then concatenated to form the 2560-bit key used in the cipher. The first 2520 bits of the key are partitioned into 45-bit subkeys. Every subkey is used in one round. Each round starts with keyed XOR, using the first 27 bits of the current round subkey, followed by a fixed permutation, a keyed permutation using the last 18 bits of the round subkey, and a fixed substitution. The process is reversed for decryption.

TUBcipher solves the problem of encrypting a small amount of data. However, open problems remain. One of them is key distribution. It is important to note that since the cryptography scheme is symmetric, the 2560-bit encryption key is the same 2560-bit decryption key used. Thus, two parties communicating using Venilia/TUBcipher must have a secure channel to establish this shared key. Another open area is the security analysis of the TUBcipher. The analysis is required to achieve confidence in the security provided by the TUBcipher.

3 TUBcipher Security Analysis

A security analysis aims at demonstrating a cryptography scheme's security. It can provide confidence in the scheme. In this case, security analysis can assist

in deciding if Venilia can become a standard or if there are better alternatives. There is no security analysis of TUBcipher. Besides, TUBcipher is based on PRINTcipher, an already established small block cipher whose security has been analyzed in the paper that introduced it [11]. However, we cannot assume that the results of this security analysis also apply to TUBcipher, as both differ in certain areas. Namely, block size, key size, number of rounds, and handling of the key in every round. PRINTcipher keeps the same key between rounds but updates it with a round-dependent value determined via a shift register-based counter. TUBcipher uses a different segment of the 2560-bit key every round, effectively using a different key every round. TUBcipher has been implemented and a security analysis has been conducted.

3.1 Implementation

The implementation of the cipher was done in Python 3 [2]. This language was chosen to keep the implementation to be as simple and straightforward as possible. To verify its correctness, we first check that given a key, an epoch, an IV, and a plaintext, the correct ciphertext is generated. The paper originally describing TUBcipher does include sample inputs and outputs [9]. Using the paper's sample key, epoch, IV, and plaintext, our implementation does generate the same ciphertext. The implementation has also been submitted 100 different randomly generated plaintexts and 100 randomly generated 2560-bit keys, allowing the test of 10,000 encryption and decryption operations. Every random plaintext was encrypted and decrypted with every random key. The plaintext before encryption and the plaintext after decryption were compared. When they were the same, the test was deemed successful. All encryption and decryption combinations were successful. The random plaintexts and random keys were generated with a random data generation procedure that is described in the sequel.

Given that several keys were needed to analyze the security of TUBcipher properly, we developed a key generation procedure to create 2560-bit keys. The procedure takes a 32-bit epoch, a 256-bit key, a 32-bit IV and creates five concatenations with the following general form:

$$Concatenation_i = (epoch||key||IV||i)$$

for $i = 0, \ldots, 4$, left padded with zeros to eight bits. Every concatenation is hashed with SHA512 and concatenated together as follows:

$$Hash_0||Hash_1||...||Hash_4.$$

This concatenation of five hashes forms the 2560-bit key. It is important to note that some parts of the key generation procedure are not addressed in the original description of Venilia/TUBciper, such as the generation of the epoch, 256-bit key, and IV. Thus, for the 256-bit key, we generate a random 256-bit string. The epoch and IV do not need to be random as long as their combination is only used once. A counter is used that is incremented after every key generation. Given that the counter is used to generate 64-bit of data, no duplicate epoch and IV

combination is generated until 1.8×10^{19} keys are produced. Hence, no duplicate epoch and IV combination has been used during the analysis of TUBcipher.

The 2560-bit keys were used to create several ciphertexts for the security analysis. Every ciphertext was created using a 2560-bit key and a 27-bit plaintext. Thus, we required a procedure to generate random plaintexts. The Java `SecureRandom` class is used. It has been well analyzed and tested [10].

3.2 Indistinguishability

The indistinguishability property for a cryptographic scheme is nice to have [3]. Indistinguishability means that for any plaintext, given the ciphertext produced by the cryptographic scheme and a random string, an adversary cannot determine which is the ciphertext with odds significantly better than 50%. In other words, no information can be gained from analyzing the ciphertext alone [14]. This property is investigated theoretically and empirically.

Definition 1 (Birthday attack). *The goal of the birthday attack is to find two plaintexts x_1 and x_2 that are encrypted with the same nonce and produce ciphertexts x_1' and x_2' such that $x_1' = x_2'$. The pair x_1, x_2 is called a collision.*

Proposition 2. *Let us consider the TUBcipher used to encrypt m one-block messages. An adversary perpetrating the birthday attack has the probability of success at most $1 - \left(\frac{2^{37}-1}{2^{37}}\right)^m$.*

Proof. Given that the IV is five bits and Epoch ID is thirty-two bits, there is a thirty-seven-bit nonce associated with every message. There are $2^{37} \cdot 2^{37} = 2^{74}$ unique nonce pairs while there are $2^{37} \cdot \left(2^{37} - 1\right)$ pairs of nonces, where the coupled nonces are different. Hence, the non-collision probability for two random one-block messages is

$$\frac{2^{37} \cdot (2^{37} - 1)}{2^{74}} = \frac{2^{37} - 1}{2^{37}}.$$

For m one-block messages, the non-collision probability is

$$\left(\frac{2^{37} - 1}{2^{37}}\right)^m.$$

Hence, the collision probability is at most

$$1 - \left(\frac{2^{37} - 1}{2^{37}}\right)^m.$$

Table 1 provides the probabilities of collision as a function of m, in logarithmic form. TUBcipher does not achieve perfect indistinguishability due to a non-negligible collision probability. But let us also investigate that empirically.

To investigate empirically, we need a metric of randomness. Entropy provides this metric. In information theory, the entropy of a random variable is the amount of information, or uncertainty, that a variable contains based on its possible

Table 1. Probability of collision versus the number of messages, in logarithmic form ($\log_2 m$).

$\log_2 m$	Collision Probability ($\times 10^{-9}$)
1	0.0146
2	0.0291
3	0.0582
4	0.116
5	0.233
6	0.466
7	0.931
8	1.86

outcomes [16]. If all outcomes have the same probability, the variable's entropy equals the theoretical maximum value, which is one. This value signifies that the outcome of the event is uncertain. The outcome is effectively random. Such a variable contains the maximum amount of information. If some outcomes are more likely than others, the value of the variable's entropy is between zero and one, but not zero or one. These values signify that the outcome of the event is not completely uncertain. The outcome is not completely random. This variable would contain less information than the theoretical maximum. The following equation defines the computation of entropy:

$$H(X) = -\sum_{i=1}^{n} P(x_i) log P(x_i) \tag{1}$$

The concept of entropy is used to measure the randomness of TUBcipher. TUBcipher encryption is the random variable X. The ciphertexts produced are possible outcomes $x_1, \ldots, x_n$, with respective probabilities $P(x_1), \ldots, P(x_n)$. When entropy $H(X)$ has the value of one or very close to one, it signifies that the ciphertexts are effectively random. In other words, such a value signifies that the ciphertexts are indistinguishable from a random string. It is demonstrated empirically that TUBcipher possesses the indistinguishability property.

To measure the entropy of generated ciphertexts, we used the pseudorandom number sequence test tool ENT authored by Walker [18]. This tool can measure the entropy of a bit string at the bit level, producing a value between zero and one. Twelve data points of 5,000 ciphertexts each were collected for five distinct data sets. The first three sets measure the entropy value of ciphertexts from the TUBcipher, while the last two sets measure the entropy of other random sources for comparison. The first set, 1PXK, measured the entropy of ciphertexts generated from the same plaintext and 5,000 different keys. The second set, XP1K, measured the entropy of ciphertexts generated from 5,000 different plaintexts and the same key. The third set, XPXK, measured the entropy of ciphertexts

generated from 71 different plaintexts and 71 different keys for a total of 5,041 ciphertexts. The fourth set measured the entropy of 5,000 random 27-bit strings generated from Java's SecureRandom class. The final set measured the entropy of 5,000 random 27-bit strings generated from the Windows `CryptGenRandom` function, being called via Python's `os.urandom()` function. Statistical measurements were made from those 12 data points, including the mean, range, variance, standard deviation, and standard error. A high entropy mean value was expected from the set of data generated from Java's `SecureRandom`. A lower entropy mean value was expected from the set of data generated from Windows `CryptGenRandom`.

Table 2. Statistical measurements of entropy from five sets of 12 data points. Each data point consists of ∼132KB of 27-bit strings.

	Mean	Range ($\times 10^{-5}$)	Variance ($\times 10^{-10}$)	Standard Deviation ($\times 10^{-5}$)	Standard Error ($\times 10^{-6}$)
1PXK	0.9999	1.1	0.103	0.321	0.926
XP1K	0.9999	2.2	0.515	0.718	2.07
XPXK	0.9999	1.9	0.684	0.827	2.39
SecureRandom	0.9991	46.1	145	12	34.8
CryptGenRandom	0.9999	1.7	0.342	0.585	16.9

A summary of the statistical measurements calculated from the gathered data of all five sets is displayed in Table 2. The range, variance, standard deviation, and standard error of the sets 1PXK, XP1K, XPXK, and CryptGenRandom are low, indicating that the size of the data sets is sufficiently big. As expected, the mean entropy value of Java's SecureRandom is close to the theoretical best value of one, with a difference of 0.000922. This is consistent with previous research [10]. Unexpectedly, the mean entropy value of Windows' CryptGenRandom is higher than that of Java's SecureRandom, with a difference of 0.000918. This could reflect changes that have been made to Windows' CryptGenRandom algorithm since 2009, which is when the cryptanalysis was performed, or it is possible that given a larger data set, the mean entropy value of Java's SecureRandom would have been higher than CryptGenRandom's. However, the mean eFor example, withpy values for each set of TUBcipher ciphertexts 1PXK, XP1K, and XPXK, are close to the theoretical maximum, with the highest difference being 0.000008.

Given that the entropy measured from ciphertexts is equivalent to or better than the entropy measured from randomly generated strings with SecureRandom and CryptGenRandom, we can be confident that an adversary would not be able to distinguish the ciphertexts from random strings with odds better than 50%. Thus, we can conclude that TUBcipher ciphertexts look nearly perfectly random

4 Venilia Key Distribution

Augmenting Venilia with key distribution is discussed. A symmetric cryptography scheme such as the TUBcipher encrypts and decrypts data with the same key. This is a secret shared by both the sending and receiving parties. Typically, symmetric keys are generated and transmitted through a secure channel before the data payload is sent. With Transport Layer Security (TLS), asymmetric cryptography schemes, such as Rivest, Shamir and Ademan (RSA), are used to securely share symmetric session keys, which are then utilized to encrypt and decrypt the data payload of the session [5]. Asymmetric ciphers make great candidates for symmetric key distribution as they use two separate keys for encryption and decryption. The key used for encryption, the public key, can be shared in open channels, while the key used for decryption, the private key, is not shared. This provides a secure channel to share secrets with the private key holder. This method works well for typical Internet applications, as an entire symmetric key can be shared within one packet. However, underwater networks have unique constraints. One is the amount of data that can be sent with one packet. Sending a 256-bit symmetric key in the clear or encrypted with an asymmetric cipher is not feasible. Given the largest data payload possible with JANUS, 34 bits, eight packets would need to be transmitted to send a 256-bit key. As such, Internet key distribution solutions are not applicable for JANUS. A new solution must be created with the underwater medium in mind. Two approaches are explored: key pre-configuration and dynamic key distribution. In the sequel, we assume the principle of confidentiality states that only the sender and receiver of a message should be able to access the information it contains [13].

4.1 Key Pre-configuration

One simple solution is configuring nodes with 256-bit symmetric encryption keys before deploying them underwater. Nodes can send encrypted Venilia packets to other nodes, given that they have their symmetric keys. However, simplicity comes with flexibility constraints. Once a network of nodes is configured and deployed, no new nodes can be added. Let us assume that we have n nodes to be deployed. Before deployment, each node is configured with its key and the other's 256-bit key. As such, every node has n keys: $\{k_1, k_2, ..., k_n\}$. To deploy a new node into the network, node $n+1$, the node would have to be configured with the keys of every other node in the network. This new node would need $n+1$ keys $\{k_1, k_2, ..., k_n, k_{n+1}\}$. Since each node currently in the network was not configured with the key for node $n+1$, no other node could communicate confidentially with it.

No new keys can be distributed to the nodes already deployed without a dynamic method for key distribution. Therefore, no further nodes can be added to the network for confidential communications. To circumvent this problem, every node could be configured with the same network key before deployment. Using a single network key allows for new nodes to be added to the confidential network since the number of keys a node must be configured with no longer

increases with the addition of nodes within the network. Suppose all confidential communications within the network are encrypted with the same key. Without confidential node-to-node communications, any network node can decrypt encrypted messages.

4.2 Dynamic Key Distribution

An alternative solution is proposed, more flexible than the solution of Sect. 4.1, at the expense of simplicity. This solution involves four components: a specific network topology, node pre-configuration, session key generation, and a session protocol.

Network Topology. Underwater networks are divided into subnetworks. Each subnetwork node shares one 256-bit master key. Each subnetwork contains regular nodes but may also have border nodes. Border nodes can communicate confidentially to at least one other node from another subnetwork. Subnetworks may also contain mobile nodes. The nodes that communicate on several subnetworks simultaneously are defined as mobile. Figure 3 pictures an example of this topology. It is important to note that this approach avoids key desynchronization issues since the master key never changes. To deal with border node failures, backup border nodes can be included to take over when they have heard the border nodes for a certain delay.

Pre-configuration. All underwater nodes must be preconfigured with the 256-bit master key of the subnetwork they belong to. Since non-border and non-mobile nodes only require a 256-bit master key, the number of keys that these nodes must know never changes and always remains at one. This allows newly deployed nodes to communicate confidentially with any other subnetwork node. In other words, new nodes can be deployed within these subnetworks without concern. Border nodes must also be preconfigured with the master keys of other subnetworks they connect to. Border nodes allow newly deployed subnetworks to communicate confidentially with other subnetworks. Already deployed border nodes within the existing subnetworks do not need to be modified, as the one deployed in the new subnetwork can be preconfigured with the required master keys instead. Mobile nodes must be preconfigured with the master keys of the subnetworks they use.

Session Key Generation. All network nodes using the same key violates the confidentiality principle. To circumvent this issue, the master key can be used to send an encrypted session key instead of the data itself. The session key, specific to a session between two nodes, can then be used to send confidential data. Since the sender and recipient only know this session key, data can only be decrypted by those two nodes, and thus, this approach provides better confidentiality than the simple solution. However, an entire 256-bit key is infeasible to send with

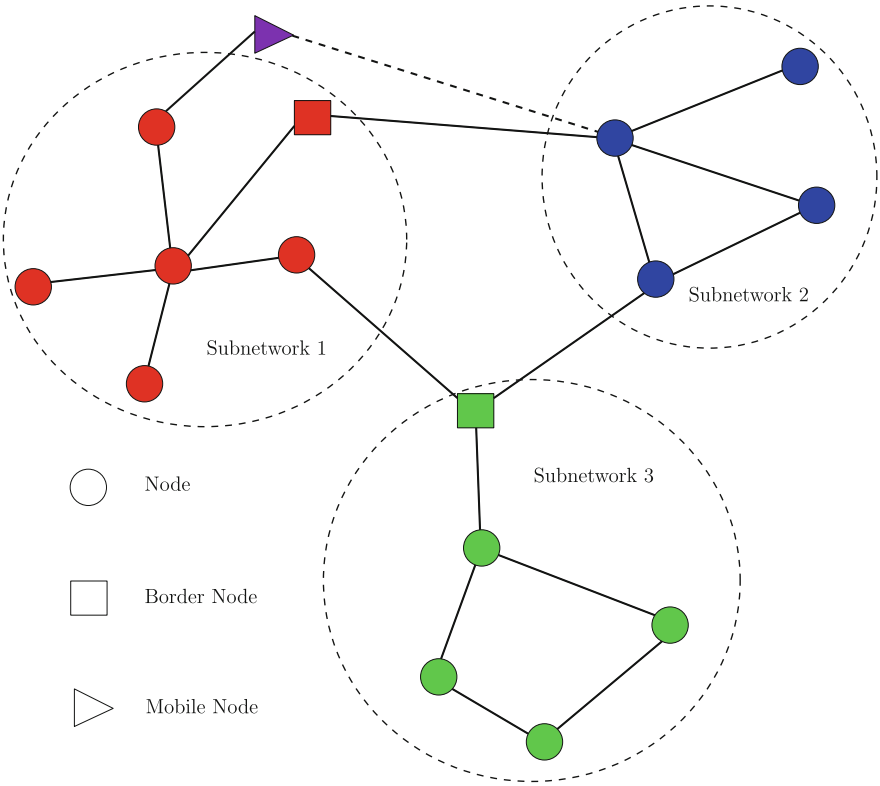

Fig. 3. Example of network topology for Venilia key distribution solution.

Venilia. Thus, a session key must instead be generated from some known secret and information that can be sent with Venilia and JANUS packets. Session keys are thus generated from the 256-bit master key, a 26-bit random number, and an 8-bit random number. Each session key is compromised of 230 bits from the master key and 26 random bits such that the session key is identical to the master key up to bit i, where bit i to bit $i+25$ are random, followed by the same bits as the master key from bit $i+26$ to bit 255. The eight-bit random number denotes the bit location to replace the master key bits with the 26 random bits. In other words, the session keys are the master key with 26 consecutive bits replaced with 26 random bits at location i to location $i+25$ where i is random. The procedure for key generation is displayed in Fig. 4.

This approach allows an entire session key to be generated from two packets; one encrypted Venilia packet sending the eight-bit random location and one clear JANUS packet sending the 26 random bits. This method is inspired by the export cipher schemes seen in the SSLv2 protocol, where a 40-bit of a 128-bit symmetric key was sent in the clear [7]. This method does indeed reduce the effective key size. However, in our case, 230 bits remain secret, and these session

$$\text{Let } x_i,\ y_i \text{ denote bit at position } i\text{:}$$

$$Master\ Key = K_m = x_0 x_1 ... x_{254} x_{255}$$

$$Key\ Piece\ = K_p = y_0 y_1 ... y_{24} y_{25}$$

$$Secret = j$$

$$\downarrow$$

$$Session\ Key = \underbrace{x_0 x_1 ... x_{j-1}}_{K_m}\ \underbrace{y_0 ... y_{25}}_{K_p}\ \underbrace{x_{j+26} ... x_{254} x_{255}}_{K_m}$$

Fig. 4. Session key generation

keys are not vulnerable to brute force attacks from outside the subnetwork. It is important to note that an attack can be perpetrated inside a subnetwork. Given that the 26-bit key piece is sent in clear and that the master key is known within the subnetwork, an eavesdropper could decrypt the eight-bit secret and build the session key. From there, they could decrypt confidential data encrypted with the session key. An eavesdropping attack would, however, have to be carried out at the start of a session to capture the important key generation information. Of course, with the assumption that legitimate nodes only share the master key, this vulnerability is no longer present. However, this solution provides a more complex system to attack than the solution where all nodes already possess the key to decrypt confidential data. As such, this solution offers better confidentiality than the solution of Sect. 4.1.

This approach is also safe from session key collisions. If the same session key is generated for two or more sessions, nodes other than the intended recipient could decrypt and read confidential data. However, given that there are 2^{26} possible random 26-bit sequences and 2^8 possible locations for these sequences, there are effective 2^{34} possible session keys. Even with a network that is compromised of one million nodes, an extreme number of nodes for an underwater network, the probability of having two nodes generate the same session key is around 0.005%.

Plaintexts encrypted with the 2560-bit extended keys generated from these session keys must also have similar entropy as plaintexts encrypted with 2560-bit extended keys generated from an entirely different 256-bit key. If the entropy of these plaintexts is not high, there might be patterns to them. Such patterns yield valuable information for malicious actors. Given that in the worst-case scenario (where the random eight-bit number is the same for two session keys), only 26 bits differ from one session key to the next, this is an important metric to verify. Following the same method described in Sect. 3.2, ciphertexts were generated from a set of random plaintexts and 256-bit keys. For our base case, these 256-bit keys were randomly generated. For our session keys, the same base 256-bit key was modified with 26 random bits at all the 256 possible locations to create the encryption session keys. In each case, 71 plaintexts were encrypted with 71 keys, generating 5,041 ciphertexts. The entropy of the 5,041 ciphertexts was then calculated with the ENT tool. This process was completed for 12 sets of

5041 ciphertexts. The results are displayed in Table 3. The entropy of plaintexts encrypted with the session keys is comparable to that of plaintexts encrypted with entirely different 256-bit keys. Given this result, it follows that the TUB-Cipher achieves indistinguishability with the session keys.

Table 3. Statistical measurements of entropy from the base case and session key ciphertexts. Measurements were taken from 12 data points, each consisting of ∼132 KB of 27-bit strings.

	Mean	Range ($\times 10^{-5}$)	Variance ($\times 10^{-10}$)	Standard Deviation ($\times 10^{-5}$)	Standard Error ($\times 10^{-6}$)
Base Case	0.9999	2.1	0.512	0.715	2.06
Session key	0.9999	1.7	0.737	0.859	2.48

Session Protocol. A protocol is defined for creating and maintaining a session between two underwater nodes. It is inspired by TLS [5]. It includes session establishment, session key generation and distribution, the transmission of confidential data, and session closure. The protocol is illustrated in Fig. 5. A node first sends an acknowledgment to another node to establish a session. The receiving node must then respond to this acknowledgment. These packets can be sent in the clear with a standard JANUS packet. The session is established once both nodes have received an acknowledgment. Once a session is established, the sending node generates the 26-bit key piece (KP) and the eight-bit secret described in the previous section. The key piece is then sent to the receiving node in the clear with a JANUS packet. The eight-bit secret is sent in a Venilia packet encrypted with the master key (MK). Using the key piece and secret; both nodes generate the session key (S_k) following the method described in the key generation section. To transmit confidential data, the sending node encrypts the data with the session key and sends the encrypted data in a Venilia packet. The receiving node decrypts the data with the same key. Confidential data is transmitted following this method until no more data needs to be sent. The session is then closed, and the session key is discarded. When confidential data needs to be transmitted again, a new session is established, and a new session key is generated.

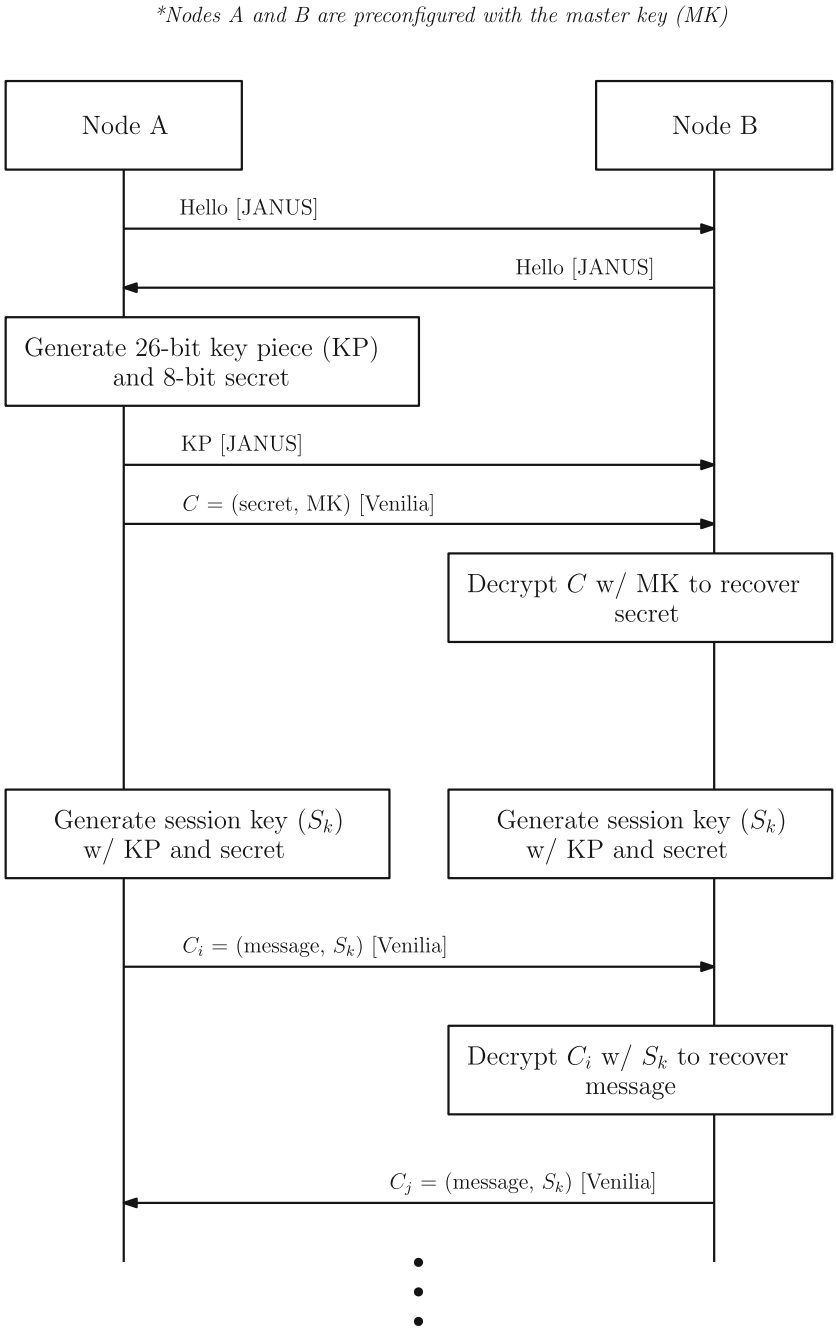

Fig. 5. Venilia key distribution and session protocol. x[JANUS] denotes message x is sent with JANUS, and x[Venilia] denotes message x is sent with Venilia.

5 Conclusion

The goal was to conduct further research for secure underwater communications. The efforts focused on developing solutions that are compatible with JANUS. As of the writing of this paper, the only solution that has been proposed to secure JANUS communications is Venilia. Efforts were devoted to improving this solution.

Areas of the Venilia class that can be improved have been highlighted. Firstly, no security analysis of the TUBcipher has been published. Without such an analysis, it is difficult to trust the security provided by the cipher. A security analysis of the cipher has been performed. The entropy of ciphertexts produced with the TUBcipher was measured. The results were very close to the highest theoretical value of one, at an average of 0.9999. It can be concluded that the TUBcipher, and by extension Venilia, provides near perfectly random ciphertexts for JANUS communications.

There was also no key distribution solution for Venilia. The eight-bit message size maximum makes key distribution challenging. However, such a solution provides important practicality. Key distribution solutions were developed for Venilia. The solutions explore practical approaches to the implementation of Venilia for underwater communications. The small message size of Venilia communications has also been considered. A codebook approach for these eight-bit messages allows 255 different messages. This could be a beneficial area of improvement. A small message size limits the flexibility of communications by only allowing predefined messages and limits potential key distribution solutions and other potentially useful security mechanisms such as cryptographic signatures. A cryptography scheme with less overhead would remedy this issue. However, such a scheme would have to be developed considering the constraints of the JANUS standard. Another way to improve message size could be to create secure communications for another underwater protocol together [1].

Acknowledgments. This work was supported by the Public Works and Government Services Canada under Contract No. W7707-227202/001/HAL through the Defence Research and Development Canada.

References

1. Barbeau, M., Blouin, S., Traboulsi, A.: Adaptable design for long range underwater communications. Wirel. Netw. 1–17 (2022). https://doi.org/10.1007/s11276-022-03027-4
2. Beaupré, Y.: TUBCipher Python Implementation. https://github.com/YBeaupre/TUBCipher.git
3. Bellare, M., Rogaway, P.: Introduction to modern cryptography (2005). https://web.cs.ucdavis.edu/rogaway/classes/227/spring05/book/main.pdf. Accessed 11 July 2022

4. Blouin, S., Lucas, C.: Early results and description of an underwater electric-field sensing and communication experiment in Bedford Basin. In: Proceedings of the 35th Canadian Conference on Electrical and Computer Engineering (CCECE2022), pp. 1–4. IEEE, September 2022
5. Dierks, T., Rescorla, E.: The transport layer security (TLS) protocol version 1.2. RFC 5246, RFC Editor, October 2008. https://www.rfc-editor.org/rfc/rfc5246
6. Gass, R., Scott, J., Diot, C.: Measurements of in-motion 802.11 networking. In: Seventh IEEE Workshop on Mobile Computing Systems & Applications (WMCSA'06 Supplement), pp. 69–74. IEEE (2005)
7. Hickman, K.: The SSL protocol. Technical report, Netscape Communications Corp (1995)
8. Hobbs A, H.S.: JANUS class 17 Venilia: secure pre-canned messaging. DSTL Cyber and Information Systems, pp. 1–22 (2021)
9. Hobbs A, H.S.: Tiny underwater block cipher (TUBcipher): 27-bit encryption scheme for JANUS class 17. DSTL Cyber and Information Systems, pp. 1–22 (2021)
10. Kenan, İ.: Security analysis of Java SecureRandom library. Eur. J. Sci. Technol. 157–160 (2021)
11. Knudsen, L., Leander, G., Poschmann, A., Robshaw, M.J.B.: PRINTcipher: a block cipher for IC-printing. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 16–32. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15031-9_2
12. National Institute of Standards and Technology: Advanced Encryption Standard (AES). https://csrc.nist.gov/publications/detail/fips/197/final. Accessed 29 June 2022
13. van Oorschot, P.C.: Computer Security and the Internet. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-83411-1
14. Phan, D.H., Pointcheval, D.: About the security of ciphers (semantic security and pseudo-random permutations). In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 182–197. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30564-4_13
15. Potter, J., Alves, J., Green, D., Zappa, G., Nissen, I., McCoy, K.: The JANUS underwater communications standard. In: 2014 Underwater Communications and Networking (UComms), pp. 1–4. IEEE (2014)
16. Rényi, A.: On measures of entropy and information. In: Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics, vol. 4, pp. 547–562. University of California Press (1961)
17. Van Walree, P.A.: Propagation and scattering effects in underwater acoustic communication channels. IEEE J. Oceanic Eng. **38**(4), 614–631 (2013)
18. Walker, J.: ENT: A pseudorandom number sequence test program (2008). https://www.fourmilab.ch/random/. Accessed 10 Dec 2021
19. Wikipedia: Replay attack. https://en.wikipedia.org/wiki/Replay_attack. Accessed 06 Oct 2022