QwyitTalk™ Unbreakable Discussion

Discussion

Version 1.3 January 2018

Author: Mr. Paul McGough, CTO, Qwyit LLC

Abstract

This paper provides a discussion of the QwyitTalk™ system security, and challenges to break the system. As a pre-requisite to this discussion, it's important to first read the QwyitTalk™ Overview, the QwyitTalk™ and Qwyit™ reference guides, as well as any and all of our papers and presentations.

Contents

QwyitTalk™ - The First Unbreakable Crypto System, Delivering The OTP Unbreakable Cipher

This document details the provable security (unbreakable) of the QwyitTalk™ Authentication and Encryption *Service* (QwyitTalk™, QT™), Security as a Service. For more on the full Qwyit protocol, and QwyitTalk™ in particular, go to www.qwyit.com.

The Unbreakable Cipher

http://www.cs.miami.edu/home/burt/learning/Csc609.051/notes/02.html

That link provides a review of the proof of a one-time-pad, an OTP. As stated in that article *"The intuition is that any message can be transformed into any cipher (of the same length) by a pad, and all transformations are equally likely."* That explains it: if any result can't be differentiated from any possibility, then it's *unbreakable.*

In order to build an unbreakable crypto*system*, one must deliver an OTP authentically to every participant, whenever they need one to communicate unbreakably with any other participant. It seems Cryptography has abandoned looking into solutions for OTP key delivery for whatever reasons – but since it's a straight line from a single use, unique key bit for every PT bit to unbreakable, if a delivery system of those keys (method and process) can be shown to meet the same unique, single-use standard, that straight line continues to an unbreakable crypto*system*.

Notes on the Qwyit Primitives for the QwyitTalk™ Cryptosystem

The Qwyit™ Protocol primitives that create the QwyitTalk™ cryptosystem are the unique PDAF, OWC, and the Combine And Extract. These provide the basis of the unbreakable cryptosystem with embedded authentication and on-the-fly OTP message keys. (Please see the Ref Docs for full descriptions of these primitives and their capabilities.)

These primitives, built with underdetermined linear equations, are the 'things' needed to turn a fast, simple linear system into an unbreakable cryptosystem:

- The ***One Way Cut*** (***OWC***) is an extremely computationally efficient one-way gate. It has been used by some of our Japanese cohorts in a few of their awarded cryptographic patents
  - The OWC one-way gate mathematically stops one getting from one equation to the next: there isn't any way to recognize either possible input from any other based on any known result, let alone the correct ones
- The ***PDAF*** (***Position Digit Algebra Function***) is an incredibly efficient PRNG
  - There is a 'philosophy' of randomness that isn't accounted for in 'traditional' mathematics (at least we're not aware of it – but it may well certainly exist): rearranging a random number, in a manner that is unsolvable, retains its randomness, whether or not it would pass some 'statistical measurement'
    - Empirically, this is: "1234" is random. Mod10 addition of an unknown, second random number "4321", generated by 'reaching into' and selecting digits from the first number using a second number or even from itself, for example leads to

"6338", which is as random as its two inputs since all of the numbers are contained in underdetermined equations

- This simple Modular addition can continually rearrange the randomness, retaining digit distributions, etc. – as long as the weakness of tending to a single digit result in every position is never allowed (i.e., IF one approaches such results, simply reseed the starting random values – we don't believe this *ever happens*, but it might)
  - This process, empirically, can be performed significantly more times than it would ever need to occur in any implementation

- The PDAF has wonderfully variant means in which to perform *Random Rearrangement*, as we'll call this property, such that it *can* be implemented to pass 'statistical measurement'
- The PDAF has an additional property of *Selective Update*, as we'll call the property of using its structure and/or values to update itself, randomly and underdetermined, in the same manner by two shared parties without any connection between them (NIL communication). The result is one-way gate protected (always underdetermined) from recovering the original, starting values (providing Perfect Forward Secrecy of every message)

- The ***Combine and Extract*** are simply the PDAF in two parts, for efficiency and performance
  - They retain the same *Random Rearrangement* property
  - They are not used for *Selective Update* only because the PDAF has more flexibility (results can be *longer* than the input, as well as operating on itself without any pointer key – see the ref docs for complete description)

QwyitTalk™ Keys

QT™ Keys have 2 parts (*Not* asynchronous):

- A part that relates to its use
- A second part that relates to changing itself (and never used through to any end result)

When the 1st part is used – the part that makes child keys for encryption – that part is an OTP. The 2nd part is never used for anything other than to swap out that 1st part – every time it is used.

Therefore, QT™ uses a secret key, in 2 parts, that changes every time it is used, and adds the other ingredients (the primitives for operations) into the process, resulting in unbreakable:

- QT™ starts with a truly random Authentication key, in 2 parts
  - Initial, secure delivery *can be accomplished*, as noted with several possible methods in our papers, where performance is not an issue (Banks still mail PINs)
- QT™ then uses that key in 2 parts completely separated by purpose
  - The *method* of separation is underdetermined with *one way gates*
- QT™ then builds truly random child Encryption keys from one Authentication key part completely (mathematically) separated by purpose
  - The *method* of separation is underdetermined with *one way gates*
- QT™ then uses the Encryption key *only* one time, on the same length Plaintext (True OTP)
- *Every key is updated after every use*
  - The *method* of update is underdetermined with *one way gates*

QwyitTalk™ is the first unbreakable Cryptosystem. It delivers a provably secure OTP, as defined since the message keys are truly random *n*-bit unique one-time-only use child keys made using underdetermined math and one way gates; and the other system keys are never used more than once.

Summary (including Protocol operation found in the Tech reference guides)

- QwyitTalk™ is an underdetermined system of equations – these are unsolvable
- QwyitTalk™ is a pass-through trust model, where keys are important, while completely unimportant and disposable (See the QwyitTalk™ Overview for this introduction/discussion)
- QwyitTalk™ provides a key management system whereby each participant has a unique OTP key that is used to talk 100% authentically and 100% securely to any other participant
- QwyitTalk™ performs this key management with two processes
    - Verified Setup Up (VSU)
    - Authentication Handshake (AH)
- QwyitTalk™ processes result in the delivery of 100% private authentication key pairs for each unique P2P communication
- The Qwyit™ Cipher delivers a 100% unbreakable OTP, using a new, mathematically underdetermined encryption bit (not a fiddled bit) for each plaintext bit

- QwyitTalk™ as a cryptosystem is Unbreakable, and delivers the unbreakable Qwyit™ OTP cipher

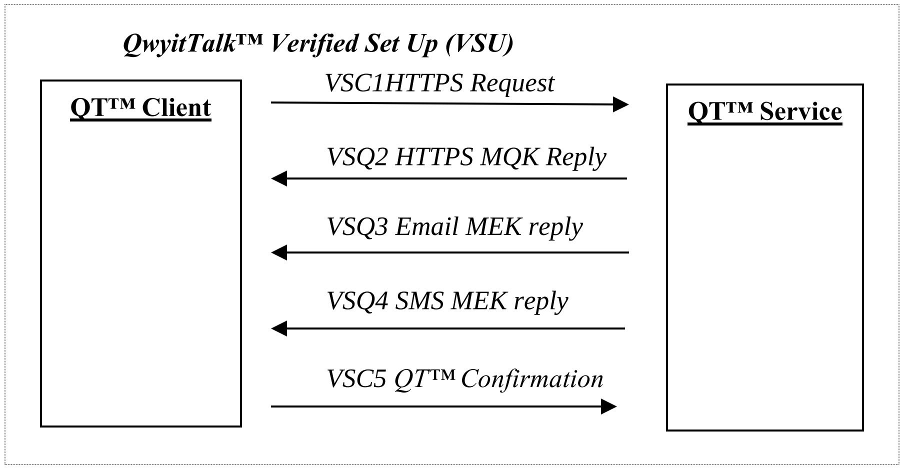
Here's QwyitTalk™ with a discussion of the three processes: Initial Key Distribution (Verified Setup, VSU), Authentication Handshake (AH) and the QT™ stream cipher (Messaging). Following are the *QwyitTalk™ Challenge*s. These deliver unbreakable proof.

QwyitTalk™ Core Processes

As a summation that can be found in detail in the Ref docs and our other papers, QwyitTalk™ is an *exact replication* of TLS; and that the processes defined by TLS are the agreed-upon, necessary information exchanges in order to authenticate and secure messaging. We're focused here on unbreakable security, but the exact benefits of QT™ over TLS in performance, efficiency and improvement are substantial – see the docs.

Initial Authentication Token/Credential Distribution (Verified Setup – VSU)

Architecture (Example method)



=

QwyitTalk™ Verified Set Up (VSU)

The VSU is like a Passport…or Driver's License…or even a Birth Certificate: they rely on each other, have some kind of process to minimize fraud and deception…aren't perfect…and generally shouldn't be assumed to be Important, but they mistakenly are.

QT™ can implement any system for Initial Key Distribution that anyone/everyone feels is 'good': HTTPS, mail, email, phone, UPS, drone, Amazon, in person, your Mother…all of these together, in combination, in sequence:

It doesn't matter how – the result is You Have A Unique Key, Known Only To You and the QT™ QDS

The *QwyitTalk™ Reference Guide* has a great deal of information on example methods for initial distribution (the VSU) of QT™ starting MQK and MEK keys – which together make up the 512-bit DSK (current recommended length.) Since it is a requirement of keeping the system secure (not unbreakable), *for this paper, and every example and Challenge in it, the assumption is the same as for that of the Unbreakable OTP: each participant has been delivered their unique key in a 100% secure manner.*

So: A VSU has happened, all the QwyitTalk™ Clients have secure unique DSKs in two parts, MQKs and MEKs, and now we're going to talk authentically and securely to each other – in any format, app, protocol, configuration (1-1, 1-many, many-1, etc.).

> ***Note:*** There are two distinct classes of interloper: active and silent. The active interloper, who we have covered already in the Intro paper; they *can* be you, but you'll be alerted, and they have to be you *again* in the *real* system, not just this public QT™ security

channel, and they can't go backwards for anything you've already done. The straightforward solution is to just perform another VSU. AND/OR – if it's a *real system* that is using a private QwyitTalk™ service as *the authentication token*, then there would *always* be a you-to-your-key 7-digit hex PIN (minimally, everyone/anyone can remember 7-values, like a phone number) that is never stored. *This* interloper's capabilities are known, minimized and solved. If the majority believe it's a good idea to implement PINs, then we'll put them in the Public QwyitTalk™.

Silent interloper: The listener, who knows every communication, from any source, at all times; except, of course, the in-person (Spy-like) key delivery methods: in person meets, drops, etc. (espionage!) Without too much difficulty, even this interloper can be defeated:

QwyitTalk™ is an underdetermined system: simply add an unknown. Since there isn't any unbreakable way to extend an unknown* in order to force The Listener to have to 'do more work' than already exists in the system; in order to thwart every attack and return the unbreakable entropy*, one simply/must add a new key*. Make the key as large as the work you want the listener to perform – if it's forever, make it *that* current length (at publication: 256-bits).

*There *is* a way to *expand* unknown work – such as having an *n-digit* unknown that points into a *z-digit* unknown number vault, 'delivering' unique results every use. This doesn't expand the 'work' of delivering an out-of-band unknown single key, but does expand the work of figuring it out: deliver a lot, point into it, work expands!

If you add a 256-bit key outside of QwyitTalk™ (un-listened), then you can use the Public System. Of course, as just noted, a private QwyitTalk™ will subvert The Listener – but if you want to keep your messaging unbreakable even to The Listener w/any particular public recipient, share another secret – sized appropriately to your particular requirements – outside the Public QwyitTalk™ system, and then you can still use it. As there is controversy whether or not government agencies put 'back doors' into crypto products, it's simple to put in a 'front door' allowing anyone using the Public QwyitTalk™ system to input 'An Outside Key' into the formulation of every *QTQS* message. If implemented and used, it will thwart all The Listeners. The war ends at Unbreakable. Privacy wins.

Next, the AH per-message key delivery and cipher unbreakable:

Per Message Token/Credential Distribution (Authentication Handshake – AH) and Messaging (QwyitTalk™Cipher)

The QT™ AH messages only include an encrypted SSK – there is no context for any decryption to result in positive knowledge of any PT. This is unbreakable; every PT is possible for every/any CT.

The only question remaining about the QT™ cryptosystem is whether the messaging ciphertexts between two participants, after receipt of their SSK, meet the unbreakable standard of an OTP:

The Qwyit™ Cipher

In either direction of The Qwyit™ Cipher, to create a single Child Key (*W*) for operation on a unique 256-bit Plaintext/Ciphertext there are:

- ***FOUR (4)*** equations
    - MOD16, Combine (which is a Mod16), Extract (a pointer selection), and XOR (or MOD16)
- ***SIX (6)*** unknowns
    - *MEK, R, MQK, A, W, PT*

For each next-Child Key, for operation on the next unique 256-bit PT/CT, there are:

- ***FOUR (4)*** equations
    - MOD16, Combine (which is a Mod16), Extract (a pointer selection), and XOR (or MOD16)
- ***SEVEN (7)*** unknowns
    - *W, R, NextR, MQK, A, NextW, PT*

Even a broken *W* resulting in a Known Plaintext attack doesn't provide any knowledge on going *forward* (as the system looks identical as it did at the start: 4 equations, 6 unknowns (the next PT)). Nor does it help in going *backward*, as one only has knowledge of two of the unknowns (*W* and *PT*), but only one equation is solved (the XOR). Retracing to the three equations that led to the XOR, one is left with 4 unknowns (*R*, *NextR* or *MEK* [depending on whether it's the first 256-bits or any succeeding 256-bits], *MQK*, and *A*) and 3 equations (MOD16, Combine and Extract). The Qwyit™ cipher is unsolvable in either direction – even with a broken Message Key. (The one thing a true OTP doesn't really include are CPA attacks – OTPs are instantly useless if you know the PT. QT™ has accommodated this 'vulnerability' with its key distribution/key creation method, rendering known PT a dead-end 'break'.

Unsolvable is ***Unbreakable***.

For examples, and if you're not convinced, work through the following Challenges.

The QwyitTalk™ Challenges

The following are four challenges that provide empirical evidence of the unbreakable properties of QwyitTalk™:

Challenge1

- The DSK updates provide mathematically separate new random versions of the starting, Authentication key (in two parts, MQK, MEK) using the feature of *Random Rearrangement* provided by the Qwyit™ primitive, PDAF

Challenge2

- Multiple different DSKs will deliver identical Ciphertext from the same Plaintext and OR salt *even through a chain of unique Message Keys* – the ciphertexts are all unique

Challenge3

- Multiple different DSKs will deliver identical Ciphertext from different Plaintexts yet same OR salt – the ciphertext is all identical
    - This *is true* for a chain of unique Message Keys (although not shown due to the ever-increasing combinatorial possibilities, even with 4-digit keys)

Challenge4

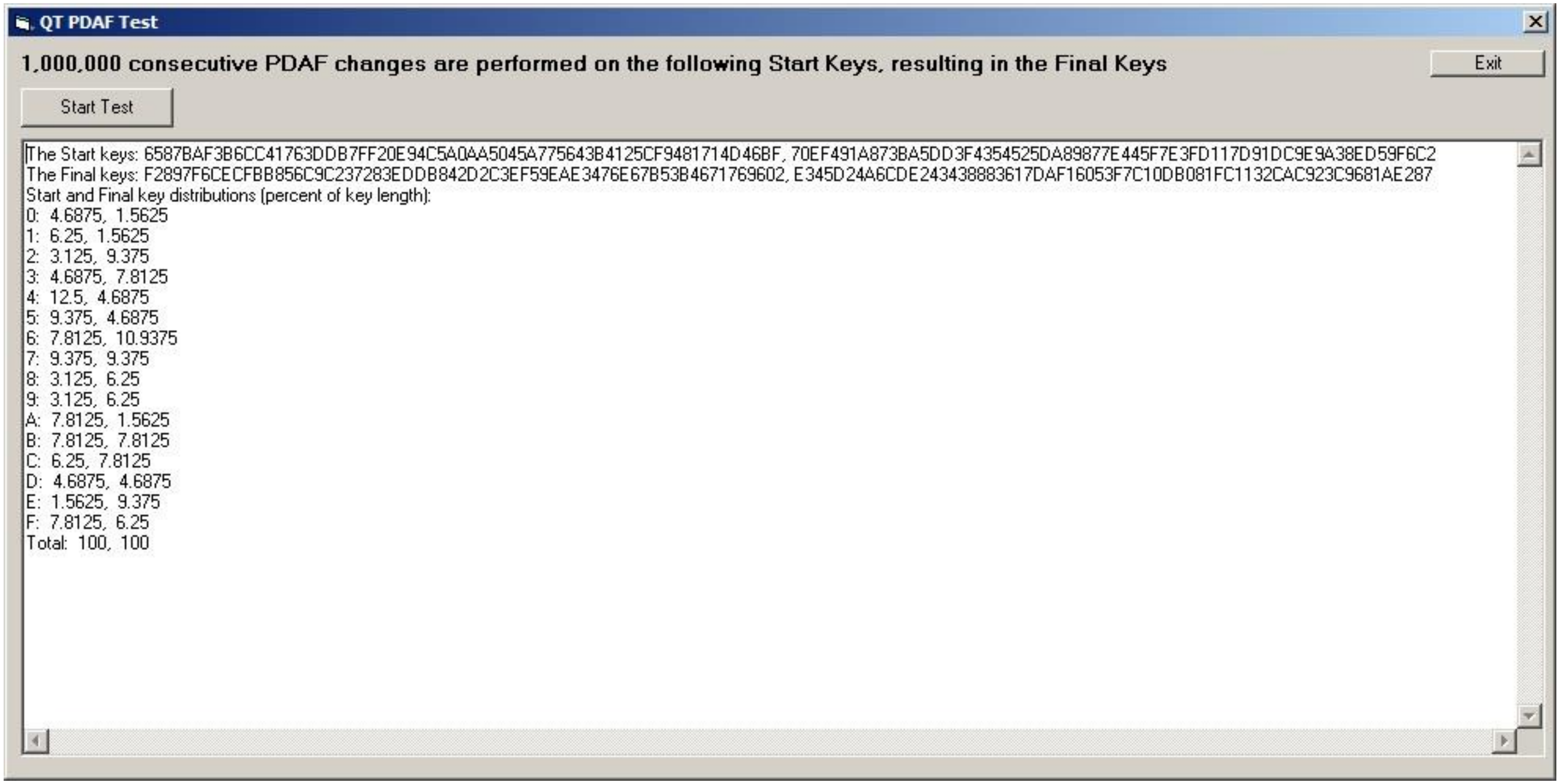
- A DSK to SSK to SMK message test with current 512-bit DSK/SSK – What are the key and plaintext results?

For all Challenges, in order to print ciphertext in this paper, all ciphertext is twice the length of the XOR byte result, where 2 hex chars (each representing the Hi/Low 4-bits of a single byte) combine to be each byte of ciphertext. Example: 6D represents the ASCII byte “m”.

Again, in summary, QwyitTalk™ is unbreakable because:

- The cryptosystem can deliver unique DSKs that result in the same ciphertext (Challenges 1 and 2)
- The Message Keys are true OTPs, delivering identical ciphertext with indistinguishable plaintext possibilities (Challenge 3)
- Challenge 4 can’t be broken *mathematically*
    - Guessing, PRNG breaks, etc. – *anything less than* presenting a method for delivering the keys and PT doesn’t apply

QwyitTalk™ Challenge #1 – Separate, Unique DSKs


QT PDAF Test
1,000,000 consecutive PDAF changes are performed on the following Start Keys, resulting in the Final Keys
Exit
Start Test

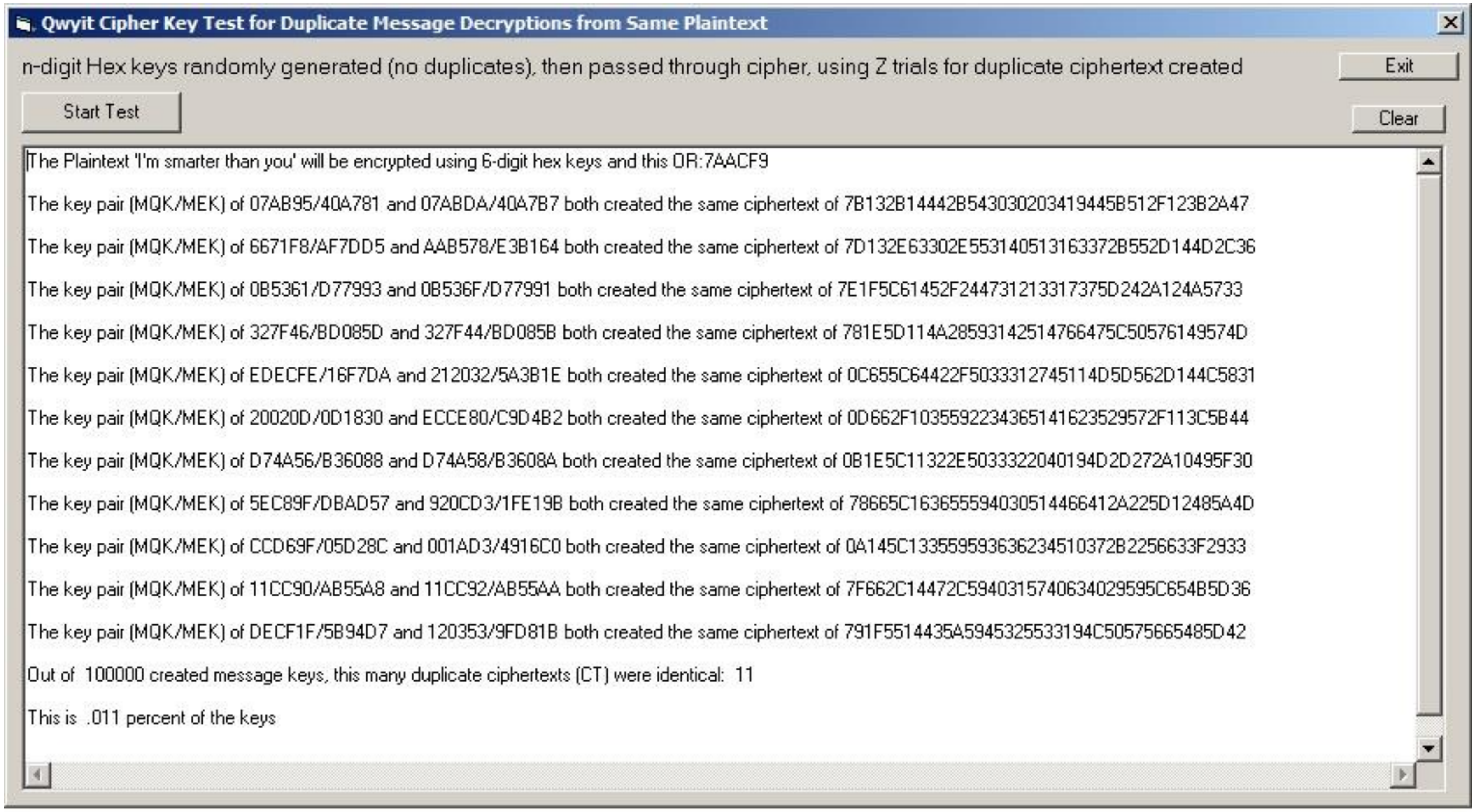
```
The Start keys: 6587BAF3B6CC41763DDB7FF20E94C5A0AA5045A775643B4125CF9481714D46BF, 70EF491A873BA5DD3F4354525DA89877E445F7E3FD117D91DC9E9A38ED59F6C2
The Final keys: F2897F6CECFBB856C9C237283EDDB842D2C3EF59EAE3476E67B53B4671769602, E345D24A6CDE243438883617DAF16053F7C10DB081FC1132CAC923C9681AE287
Start and Final key distributions (percent of key length):
0:  4.6875,  1.5625
1:  6.25,  1.5625
2:  3.125,  9.375
3:  4.6875,  7.8125
4:  12.5,  4.6875
5:  9.375,  4.6875
6:  7.8125,  10.9375
7:  9.375,  9.375
8:  3.125,  6.25
9:  3.125,  6.25
A:  7.8125,  1.5625
B:  7.8125,  7.8125
C:  6.25,  7.8125
D:  4.6875,  4.6875
E:  1.5625,  9.375
F:  7.8125,  6.25
Total:  100,  100
```

The above is a screen capture of a small program written to perform 1M PDAF *Random Rearrangements (RR)*; starting with PRNG random keys, and ending with random keys. 1M 'ephemeral' key updates in this manner (or double-length extension along with an OWC) provides a truly one-way gate; and have a long-lasting system capability, limiting the number of times a new key VSU delivery is 'required'.

Challenge1: Demonstrate, show, and prove

- A PDAF is not a one-way gate – that one can find the starting keys in less space than brute force from a known result
  - Be sure to familiarize yourself with the full capability of the PDAF function – starting offsets, pointing modes, cycle numbers, skip values, etc. *Whatever 'less' than brute force may exist can more than likely be recovered*
  - Be certain to include discovery of any reduction in strength in terms of the doubling+OWC option
- A key that passes random 'qualification' (however defined) is any less random after *n* PDAF RRs
  - And how such a 'loss' results in a system vulnerability (since the upper level keys never encrypt anything but other keys)
- Any other issues w/the PDAF primitive (and in combination w/an OWC) for RR *new, next use* (as an OTP)

QwyitTalk™ Challenge #2 – Identical unique CT, Same PT


Qwyit Cipher Key Test for Duplicate Message Decryptions from Same Plaintext
n-digit Hex keys randomly generated (no duplicates), then passed through cipher, using Z trials for duplicate ciphertext created
Exit
Start Test
Clear

```
The Plaintext 'I'm smarter than you' will be encrypted using 6-digit hex keys and this OR:7AACF9

The key pair (MQK/MEK) of 07AB95/40A781 and 07ABDA/40A7B7 both created the same ciphertext of 7B132B14442B543030203419445B512F123B2A47

The key pair (MQK/MEK) of 6671F8/AF7DD5 and AAB578/E3B164 both created the same ciphertext of 7D132E63302E55314051316372B552D144D2C36

The key pair (MQK/MEK) of 0B5361/D77993 and 0B536F/D77991 both created the same ciphertext of 7E1F5C61452F244731213317375D242A124A5733

The key pair (MQK/MEK) of 327F46/BD085D and 327F44/BD085B both created the same ciphertext of 781E5D114A2859314251476647 5C50576149574D

The key pair (MQK/MEK) of EDECFE/16F7DA and 212032/5A3B1E both created the same ciphertext of 0C655C64422F5033312745114D5D562D144C5831

The key pair (MQK/MEK) of 20020D/0D1830 and ECCE80/C9D4B2 both created the same ciphertext of 0D662F103559223436514162352957 2F113C5B44

The key pair (MQK/MEK) of D74A56/B36088 and D74A58/B3608A both created the same ciphertext of 0B1E5C11322E503332204019 4D2D272A10495F30

The key pair (MQK/MEK) of 5EC89F/DBAD57 and 920CD3/1FE19B both created the same ciphertext of 78665C163655594030514466412A225D12485A4D

The key pair (MQK/MEK) of CCD69F/05D28C and 001AD3/4916C0 both created the same ciphertext of 0A145C13355959363623451037 2B2256633F2933

The key pair (MQK/MEK) of 11CC90/AB55A8 and 11CC92/AB55AA both created the same ciphertext of 7F662C14472C5940315740634029595C654B5D36

The key pair (MQK/MEK) of DECF1F/5B94D7 and 120353/9FD81B both created the same ciphertext of 791F5514435A5945325533194C50575665485D42

Out of  100000 created message keys, this many duplicate ciphertexts (CT) were identical:  11

This is  .011 percent of the keys
```

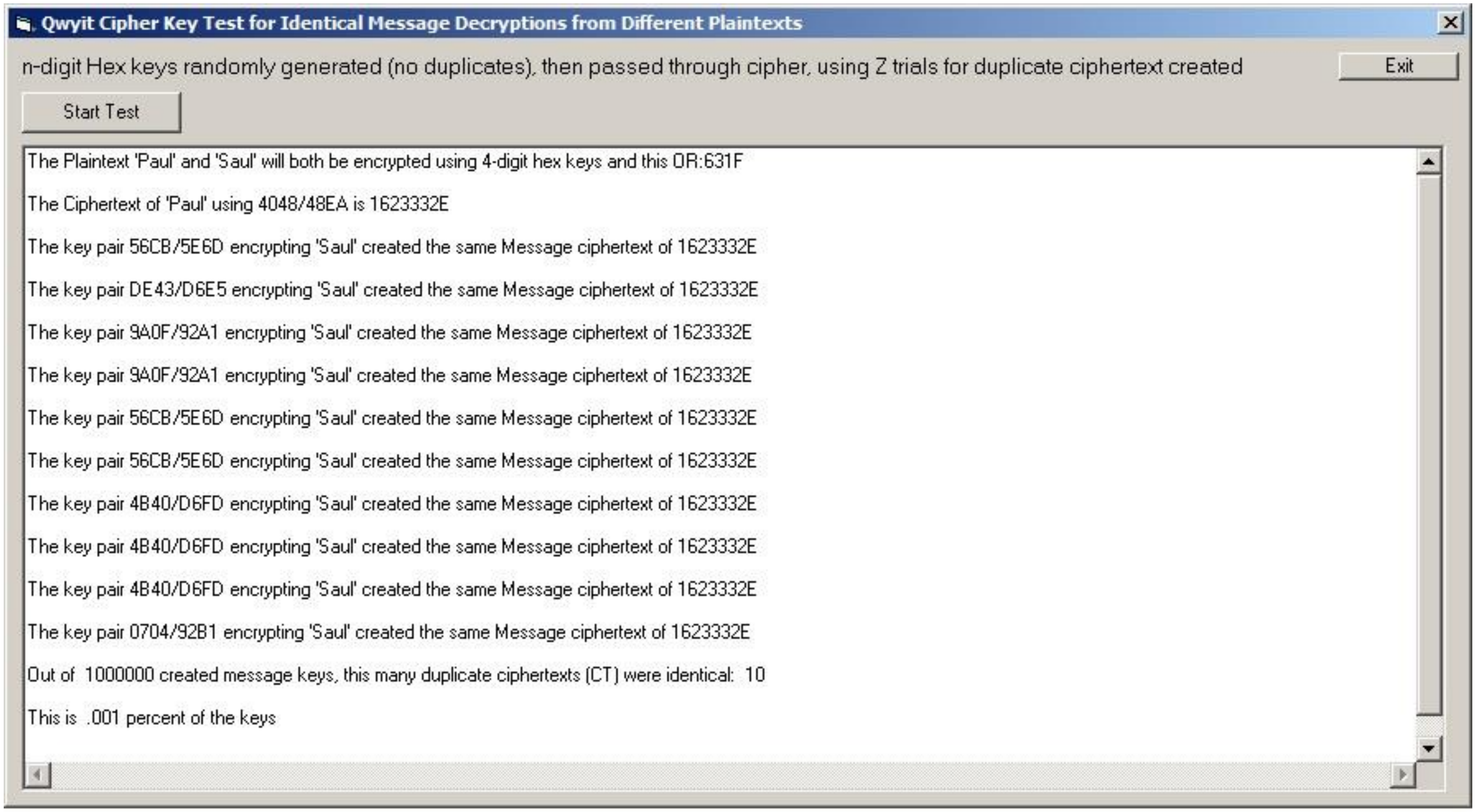
The above screen capture is from a small program written to encrypt the exact same PT (“I’m smarter than you” – sorry, that’s tongue-in-cheek!) and the same OR (“7AACF9”). This shows that multiple different DSKs will deliver identical Ciphertext from the same Plaintext and OR salt *even through a chain of unique Message Keys* (in the above, the 6-digit W’s are used in 4 re-configurations (using *NextR*) over the 20-digit PT) – and that the ciphertexts are all unique. This is the same as not ever being sure what Pad encrypted ‘Attack at dawn’, as the PT is the same, but the CT would be different.

You'll notice I used small, 6-digit hex keys so that you can easily verify these results yourself – they were encrypted using the Qwyit™ cipher substituting a PDAF/OWC combination for the Combine/Extract (since the C/E function is more 'complex' in code when the key length is shorter than the key base; and I simply didn't want to write that, since I already had the PDAF 'going around the edges'). You'll also notice that, because of the smaller-than-base length keys, some of the key pairs are almost identical (ex.: 11CC90/AB55A8 and AACC92/AB55AA only having 1 digit different in each pair), and some are quite different. You won't find that the case once the key-length is longer than the base, as the Qwyit primitives will deliver all possible results when the matrices are full (i.e., when all of the selection slots have values).

Challenge2: Demonstrate, show, and prove

- That although these are obvious, empirical examples of the underdetermined property of the Qwyit™ key processing, they are somehow solvable such that you could determine which key pair was correct (not just valid).
- That somehow, using proper, current key lengths (256-bits at publication) will result in 'easy' determination of a valid key pair
    - Some knowledgeable and motivated person may well want to present the combinatorial probabilities of the varying valid key space – in relation to the starting space and in relation to the work required to find the first one
    - Also determining a way to present how many *NextR* iterations any type/configuration of the starting DSKs any particular valid set will contain/provide. The above small example has 11 key pairs appearing out of 100K random starting DSKs (where there's 16^{12} possible combinations) resulting *through 4 iterations*. How many would last more, how many less?
        - IF you went to all of this work – one can ***only apply it to Known PT*** – see the next Challenge as to why
            - And…what does Known PT get you in Qwyit™? (Nothing, we're pretty sure…)

QwyitTalk™ Challenge #3 – Identical same CT, Different PT


Qwyit Cipher Key Test for Identical Message Decryptions from Different Plaintexts
n-digit Hex keys randomly generated (no duplicates), then passed through cipher, using Z trials for duplicate ciphertext created
Exit
Start Test

```
The Plaintext 'Paul' and 'Saul' will both be encrypted using 4-digit hex keys and this OR:631F
The Ciphertext of 'Paul' using 4048/48EA is 1623332E
The key pair 56CB/5E6D encrypting 'Saul' created the same Message ciphertext of 1623332E
The key pair DE43/D6E5 encrypting 'Saul' created the same Message ciphertext of 1623332E
The key pair 9A0F/92A1 encrypting 'Saul' created the same Message ciphertext of 1623332E
The key pair 9A0F/92A1 encrypting 'Saul' created the same Message ciphertext of 1623332E
The key pair 56CB/5E6D encrypting 'Saul' created the same Message ciphertext of 1623332E
The key pair 56CB/5E6D encrypting 'Saul' created the same Message ciphertext of 1623332E
The key pair 4B40/D6FD encrypting 'Saul' created the same Message ciphertext of 1623332E
The key pair 4B40/D6FD encrypting 'Saul' created the same Message ciphertext of 1623332E
The key pair 4B40/D6FD encrypting 'Saul' created the same Message ciphertext of 1623332E
The key pair 0704/92B1 encrypting 'Saul' created the same Message ciphertext of 1623332E
Out of  1000000 created message keys, this many duplicate ciphertexts (CT) were identical:  10
This is  .001 percent of the keys
```

The above screen capture is from a small program written to encrypt two different PTs (“Paul” and “Saul”) using the same OR (“631F”) and resulting in different DSKs creating the exact same CT (this is the “Attack at Dawn” vs “Attack at Dusk” classic OTP dilemma). Multiple different DSKs will deliver identical Ciphertext from different Plaintexts using the same OR salt – and the ciphertext is all identical. In case you’re wondering why there are the same key pairs listed multiple times (it says it found 10, but there’s really only 4), it’s because the easiest way to test this is to fix one PT/CT/OR result, and scan the other for matches – and I just didn’t bother to put in code to check the result against any I'd already found (too slow, going through both sides) I also left that in there to make a point: there are 16^4 possible CTs for each PT – but there are 16^8 possible DSKs making them – and from the last challenge, these will result in *different* CT for the same PT *on each side*. I only captured *one* CT from one side, testing against only 100K random tests dipping into the 16^8 possibilities from the other side (of which there are only 16^4 unique results, but those are also duplicated)…SO: how many times does this happen across all those combinations? *Enough to be Unbreakable*. The accurate combinatorial possibilities can be described.

Challenge: Demonstrate, show, and prove

- That although these are obvious, empirical examples of the underdetermined property of the Qwyit™ key processing, they are somehow solvable
- Somehow show that this is a problem – even though it matches the exact OTP classical proof

QwyitTalk™ Challenge #4 – A Message Test

A DSK to SSK to SMK message test with current 512-bit DSK/SSK – What are the key and plaintext results?

Here is the complete output: (Created from the *QDS-Full-Example.exe QT™ Reference Program*, available from Qwyit LLC)

During the AH, in order for Client1 to securely communicate with Client2, in response to Client1's *AHC1* request,
the *QwyitTalk™ Security as a Service* QDS sends this *AHQ3* message to Client 1, and then Client1-to-Client2 messaging ensues:

- All QT™ missing keys and PT shown redacted ([redacted])
  - The message traffic that would be sent openly is highlighted green, the commenting is informationally provided to explain the processing, as performed by the QDS and QT™ clients

QDS

Now, I'll send out the AHQ3 to just Client1 with Client2's wrapped SSK

I need to get an OpenReturn in order to encrypt the AHQ3 for Client1
OpenReturn = 7F14FCCF88D221A6C6BE5CC4922722339644F0B75D19262AD324CA6ED712276A

Click Continue in Client1...

Only AHQ3 is to Client1...
The AHQ3 Client1 message key is [redacted]

The AHQ3 first ciphertext is
71750104040C710B030277700776767 20F0801727406030779050270040075060B0471090F77717B0D777E0607020F7A02730C0E09010802
087172757774070C02020006030C070D77017B7104760402077D01770170707C0971000305727E770075767C010A720F06770C0F0E030B7D
0504747307077F77080B747707030177

The AHQ3 Client2 message key is [redacted]

The AHQ3 second ciphertext is
040277047B7D7C7507060D017672047C7B020D027A767173737100747104017 10A087777717775010C07000871060F010E747D0C0874060
072040904777 6037B777576067C7D0A737305010075727 60C73770D070F00020803050207 70760A00017970027F0A7675070772017807 0B0
609030571067271757 27E0F0607010500

[AHQ3: 0123456789ABCDEF, 7F14FCCF88D221A6C6BE5CC4922722339644F0B75D19262AD324CA6ED712276A,
71750104040C710B0302777007767 6720F0801727406030779050270040075060B0471090F77717B0D777E0607020F7A02730C0E09010802
087172757774070C02020006030C070D77017B7104760402077D01770170707C0971000305727E770075767C010A720F06770C0F0E030B7D
0504747307077F77080B747707030177,040277047B7D7C7507060D017672047C7B020D027A767173737100747104017 10A0877777177750
10C07000871060F010E747D0C08740600720409047776037B777576067C7D0A7373050100757 2760C73770D070F0002080305020770760A0
0017970027F0A76750707720178070B0609030571067271757 27E0F0607010500] sent using Port 4180 to Client1

AHQ3 processing now switches to Client1 and Client2

Click Continue in Client1...

Client1

Now I am going to talk to Client2 in our application...

First, I'll create a Qwyit Return (QR) value
QR is
DFEB16125CF4510FCC5091500D312F93773EF72D2AFFFE9AC3B40A5FC2AAEA04F7861E1E250E31337E6DACD7E9FEA5A0E974613DB8B10E2
32B0C6C03CFBF75AE

Click Continue in Client1...

Next, I'll use the SSK from the QDS and the just created QR to create a Session Master Key (SMK, in 2 halves, SQK and SEK)
New Client1-Client2 SQK is [redacted]
New Client1-Client2 SEK is [redacted]
Click Continue in Client1...

Now ready the message, which will be: [redacted]

Then, encrypt it using QwyitTalk...

OpenReturn for this message is B57BB6519D3077EC0A11B38AB3198183AF7E981AC9B3A87589D345E60CB9F8B1

The QTQS message key is [redacted]

QwyitTalk Qwyit Start (QTQS) is
0123456789ABCDEF,
DFEB16125CF4510FCC5091500D312F93773EF72D2AFFFE9AC3B40A5FC2AAEA04F7861E1E250E31337E6DACD7E9FEA5A0E974613DB8B10E2
32B0C6C03CFBF75AE, B57BB6519D3077EC0A11B38AB3198183AF7E981AC9B3A87589D345E60CB9F8B1,
7F14FCCF88D221A6C6BE5CC4922722339644F0B75D19262AD324CA6ED712276A,040277047B7D7C7507060D017672047C7B020D027A7671
73737100747104017 10A087777717775010C07000871060F010E747D0C087406007204090477760 37B777576067C7D0A7373050100757276
0C73770D070F00020803050207707 60A00017970027F0A76750707720178070B0609030571067271757 27E0F0607010500,72527544495911
6B7B66590641654811755B5F5E41070867415A5A386B7F29155617010933737767 2B08647B000238304250 75246B76532D02080140425C76
4C

Now, QT processing continues in Client 2...

Click Continue in Client1...

Client2

I've just received a QTQS message from Client1...I'll process it!
First, I need to unwrap the SSK...

The Client2 AHQ3 received message key is: [redacted]
The Client2 AHQ3 received plaintext, which is the SSK, is:
[redacted]

Next, I need to unwrap the SMK...

Now, take the just received QR and create the SMK for this message

New Client1-Client2 SQK is [redacted]
New Client1-Client2 SEK is [redacted]

Click Continue in Client1...

The QTQS message key is [redacted]

QTQS plaintext message is [redacted]

Now I'll reply back to Client1...message is: [redacted]

Click Continue in Client1...

OpenReturn for this message is 6843989379FC5A8E7AB710F95D6D0B3723ACEBB3F0F0BBDE367008D45E2F9644

The QTQT message key is [redacted]

QwyitTalk Qwyit Talk (QTQT) is CBEC3BF4B47CA8B0, 6843989379FC5A8E7AB710F95D6D0B3723ACEBB3F0F0BBDE367008D45E2F9644, 305C17750E7D354A6B137532755E6B330F5E5B572C7D50025E0F1C54447D1C667D5F2702436F6A030053670B4E577846517C241C2C5A076A5C462B2D75722D36

Now, QT processing continues in Client 1...

Click Continue in Client1...

Client1

I've just received a QTQT message from Client2...I'll process it using our shared SMK keys and the sent Open Return!

The QTQT message key is [redacted]

QTQT plaintext message from Client 2 is [redacted]

Click Continue in Client1...

After sending and receiving and the SMK key life is reached, both Clients perform a PDAF PFS ephemeral DSK key update

This is accomplished using the SQK and SEK in a PDAF. Optionally, for a 1,024-bit length, and then performing an OWC to pare to 512-bits
First, here in Client 1...

Click Continue in Client1...

Here is the PDAF result for Client1 new SQK/SEK keys:
[redacted]

Here is OWC final result for Client1 new SQK key: [redacted]
Here is OWC final result for Client1 new SEK key: [redacted]

NOTE: That second key looks like it has more letters in it than numbers…OOOOHHH NOOOOO!!!!...leaking entropy…☺

Now, update the SMK in Client 2...

Click Continue in Client1...

Client2

Here is the PDAF result for Client2 new SQK/SEK keys:
[redacted]

Here is OWC final result for Client2 new SQK key: [redacted]
Here is OWC final result for Client2 new SEK key: [redacted]

Click Continue in Client1…

This concludes the AH and QwyitTalk demonstration. Thank you!

Client1
This concludes the AH and QwyitTalk demonstration. Thank you!

THIS COMPLETES CHALLENGE #4

Challenge: Demonstrate, show, and prove

- The Answer – keys and PT
- Provide rational response to the real question: *Whether or not you agree with us on the full capabilities of QT™, the performance is unquestionably superior (see 2015 NIST Lightweight Cryptography benchmark submission), and its potential is substantial – and it is unbreakable… why on earth aren't you using it somehow/somewhere?*

Afterward

We think it's important, just as a recap, to state that, unequivocally, there are absolutely no new cryptographic 'processes' in QwyitTalk™ that haven't been invented; and, of course, that's the whole point: QT™ *works just like all of the well-known, well-defined, well-researched, well-thought out, well-attacked crypto best-practices:*

- QwyitTalk™ Security as a Service is TLS
- The Qwyit™ Cipher works just like any other block/stream cipher
  - Random IV (OR)
  - Key mixing to derive child keys
  - Child keys perform quick-step encryption (XOR, etc.)
  - Embedded authentication (called Additional Authenticated Data (AAD) and other terms)

The result is something quite different, though, than the inputs: QT™ is unbreakable.

References

www.qwyit.com

All of the pertinent papers are available there – and references are included.

Epilogue

We'd like to mention all of the *real* cryptographers that helped us make QT™ the complete system it is today:

Dr. Alan Sherman, UMBC, whose early critique forced us to complete the full protocol, matching the system's properties to our claims
Dr. Hatsukazu Tanaka, Kobe Institute of Computing, whose acceptance and promotion of our work was given its highest praise by including it in his own – extending his research on key expansion and ciphers
Dr. Giovanni Di Crescenzo, NYU Tandon School of Engineering, whose full, in-depth independent reviews – and NIST submission/work – and constant questioning of our work, led us here – completion

All truth passes through three stages. First, it is ridiculed. Second, it is violently opposed.

Third, it is accepted as being self-evident. - Arthur Schopenhauer