Reference implementation of high-security (eg. pure ciphertext transmission, secret-sharing, third-party prevention of rubber-hose cryptanalysis of hard disk, etc).

Usage

commKey

```
./commKey _setup
```
(run only once to generate directories and some symmetric keys)

```
echo -n BASH_echo PASS | ./commKey _query_search

_set_commFields_anon 3w8iU0eBs7pGaMgxlc4vsgyi
echo -n PASS | _commKey_tx "3w8iU0eBs7pGaMgxlc4vsgyi" | <ssh user@server commKey _commKey_interpreter> | _commKey_rx "3w8iU0eBs7pGaMgxlc4vsgyi"
echo -n PASS | _commKey_tx "3w8iU0eBs7pGaMgxlc4vsgyi" | <ssh user@server commKey _commKey_interpreter> | _commKey_search

echo -n PASS | ./commKey _client_commKey "3w8iU0eBs7pGaMgxlc4vsgyi" <ssh user@server commKey _commKey_interpreter>

./commKey _client_commKey_generate "" <ssh user@server commKey _commKey_interpreter>
./commKey _client_commKey_generate "" <../testServer/commKey>



#All commands defined under "_prog/composite/interpreter/_interpreter_filter.sh" .

echo -n BASH_echo PASS | ./commKey _query_diag
echo -n PASS | ./commKey _query_diag

#STORE, 24char filename, "PASS" data.
echo -n STRO_$(_uid 24)PASS | ./commKey _query_interpreter

echo -n LSFS | ./commKey _query_interpreter
echo -n SWPD | ./commKey _query_interpreter

echo -n LOGA | ./commKey _query_interpreter
echo -n LOGR | ./commKey _query_interpreter
echo -n LOGS | ./commKey _query_interpreter

echo LOGO | ./commKey _query_diag

echo -n NEWU | ./commKey _query_interpreter
#3w8iU0eBs7pGaMgxlc4vsgyi
echo -n GKAU_3w8iU0eBs7pGaMgxlc4vsgyi | ./commKey _query_interpreter
echo -n RDYU_3w8iU0eBs7pGaMgxlc4vsgyi | ./commKey _query_interpreter
```

fragKey

```
#Keys will be created in "./smoke/project" directory or similar. To be moved to "./temple/_fragKey/portal" directory or similar. Do NOT move all keys to unencrypted storage.


./fragKey _volatile

./fragKey _purge_fragKey_header
./fragKey _purge_fragKey_container
./fragKey _generate_fragKey_header

#REQUIRED. If "$containerPath" is to be overloaded by an existing file (eg. device) this command MUST be run before setting the variable.
./fragKey _generate_fragKey_container <bytes>

./fragKey _generate_fragKey_direct
./fragKey _deploy_fragKey_direct #<keyID>)
#Key must be moved, scripted, or deleted.

./fragKey _generate_fragKey_special
```

```
./fragKey _deploy_fragKey_special #<keyID>)
#Key must be moved, scripted, or deleted.

./fragKey _generate_fragKey_distributed
./fragKey _deploy_fragKey_distributed #<keyID>)
#Key must be moved, scripted, or deleted.

./fragKey _generate_fragKey_flexible
./fragKey _deploy_fragKey_flexible #<keyID>)
#Key must be moved, scripted, or deleted.

./fragKey _purge_fragKey_header
./fragKey _recover_fragKey_direct #<keyID>

./fragKey _purge_fragKey_header
./fragKey _recover_fragKey_special #<keyID>

./fragKey _purge_fragKey_header
./fragKey _recover_fragKey_distributed #<keyID>

./fragKey _purge_fragKey_header
./fragKey _recover_fragKey_flexible #<keyID>


./fragKey _create_container
./fragKey _format_container
./fragKey _mount_container
./fragKey _unmount_container
./fragKey _remove_container



./fragKey _purge_fragKey_header
./fragKey _remove_container

#Defined by "ops", see "_local/ops" example.
./fragKey _grab



./fragKey _password_fragKey #<keyID>
```

Design

Outstanding Resistance Authentication & Confidentiality Lightweight Encryption (ORACLE) protocols. Pure ciphertext communications, and other security functions. Compatible with widely used cryptography software, and thus, highly portable.

ORACLE may perhaps have a several years long operational history, perhaps multiple revisions, perhaps for personal use, perhaps not released, perhaps still not entirely released.

Use case of ORACLE to encrypt laptop storage has perhaps become less important to any who may have perhaps authored ORACLE, and thus perhaps outlived a need for either extensive code review or secrecy. Other developers, perhaps unanticipated and unknown, may perhaps need such high-security software, as politics may perhaps be continuing to remain hostile to developers technical needs for privacy of their own thoughts, notes, and related resources.

Protocol

Symmetric

*) Short list of keys (including counter) already exchanged.
*) Transmitter encrypts plaintext (Data), signs ciphertext with MAC (Authentication), increments counter (Counter), adds parity to resulting ciphertext/authentiation, encrypts (Parity), modulates ciphertext/authentication into analog channels.
*) Receiver demodulates all analog channels, decrypts all channels with all keys, searches all streams for valid ciphertext/authentication.
*) Receiver checks counter, rejects ciphertext replay.

Speak to when spoken to. Client transmitter always increments counter. Server receiver may only reply, and may not initiatite transmission. Paging or bulk transfer, any broadcast transmissions directly to client, of any timing or type, are scheduled by prior client/server messaging.

Safety

High-security and reliability offered by symmetric pure-ciphertext authenticated encryption can be undone by a single incident of misuse. Do NOT expect security without preparation, redundancy, contingencies, rehearsal, careful deployment, and expertise. Do NOT expect security will be guaranteed, especially with the underlying vulnerabilities of existing computer hardware and software.

Code review may not yet be adequate. Exploits, misused cryptographic primitives, etc, are NOT absolutely guaranteed absent.

Regard this as a rough prototype for now. If you do use this for security, best practice may be to ensure other security and redundancy layers are in place (eg. SSH).

Specific vulnerabilities already apply to some use cases.

*) XTS mode of operation has a distinguishing attack (reduction to ECB-like statistical patterning) probability of <1/2^53 when a single key has been used to encrypt >2^40 blocks, reportedly >1TB. Risk is significant at >1PB. See article "Follow-up on NIST's Consideration of XTS-AES".
*) Host CPU, OS, and any running software, will introduce vulnerabilities, which should reasonably be expected to have zero-day exploit vulnerabilities known to and unreported by many.
*) MSW compatibility is theoretically technically very achievable but for security should NOT ever be a reasonable question, as MSW is too much far less secure than this or other security software. There is simply no reasonable use case for the high-security of ORACLE in which MSW is a host or guest OS anywhere with access to a CPU with access to the same filesystem. Android and iOS are usually used with hardware that is not much better and have their own security. If MSW use is absolutely necessary, it may be better to at least remotely access limited services from a physically separate Linux/UNIX computer.

In any case, the high-security which would make ORACLE worthwhile WILL be compromised unless no exploitable OS is present, and multiple keys are used across multiple relatively secure CPU architectures to reduce the possibility of a single exploit compromising desired security. Some ARM and FPGA CPU cores may be necessary.

Functions to cause dependency on multiple different systems to decrypt a single system (ie. combining multiple keys from multiple places into a single keyfile useless if incomplete) are included. Use those functions with sufficiently secure and disparate hardware if the high-security is offered is actually important.

Data loss is also highly possible if ORACLE software is used incorrectly. In particular, if a container is copied without changing 'containerid', both filesystems may be incorrectly mounted, reformatted, deleted, etc, unintentionally.

You have been warned. Your accident is your accident.

commKey

```
./commKey _setup
./commKey _client_commKey_generate_test
```

fragKey

```
./fragKey _setup
echo -n testtesttest | ./fragKey _passEncrypt ./test ./testOut
echo -n testtesttest | ./fragKey _passDecrypt ./test ./testOut

#A unique "containerid" is assigned, and must be reset if container is copied. After copying, at the new container file directory, use a command similar to the following.
_uid 26 > ./temple/_fragKey/containerid
```

Reference

https://en.wikipedia.org/wiki/EARN_IT_Act#Legislative_history
	'reintroduced by Lindsey Graham and Richard Blumenthal in February 2022[16] and passed unanimously by the Senate Judiciary Committee.[17]'
	-
	'former Attorney General Barr has extensively argued that the use of end-to-end encryption by online services can obstruct investigations by law enforcement, especially those involving child exploitation and has pushed for a governmental backdoor into encryption services'
	'The Senators behind EARN IT have stated that there is no intent to bring any such encryption backdoors with this legislation.[29] However, according to The Washington Post, Richard Blumenthal said that "lawmakers wouldn't offer a blanket exemption to using encryption as evidence, arguing companies might use it as a 'get-out-of-jail-free card.'"[30][31]'
	-
	'As an implicit response to EARN IT, Wyden along with House Representative Anna G. Eshoo proposed a new bill, the Invest in Child Safety Act, in May 2020 that would give US$5 billion to the Department of Justice to give additional manpower and tools to enable them to address child exploitation directly rather than to rely on technology companies to rein in the problem.[36]'

https://www.washingtonpost.com/politics/2022/02/23/most-cyber-pros-give-thumbs-down-earn-it-act/

https://hackaday.com/2020/06/16/disable-intels-backdoor-on-modern-hardware/
https://hackaday.com/tag/intel-management-engine/
	'something horrible'
	'Intel's Management Engine (ME) is a completely separate computing environment running on Intel chipsets that has access to everything.'
	'When - not 'if' - the ME is finally cracked open, every computer running on a recent Intel chip will have a huge security and privacy issue. Intel's Management Engine is the single most dangerous piece of computer hardware ever created.'

https://redmine.replicant.us/projects/replicant/wiki/SamsungGalaxyBackdoor
https://www.pcworld.idg.com.au/article/540473/some_samsung_galaxy_devices_contain_file_access_backdoor_replicant_developers_say/
https://www.kaspersky.com/blog/popular-samsung-devices-allegedly-contain-backdoor-2/4113/
https://en.wikipedia.org/wiki/Replicant_(operating_system)

https://en.wikipedia.org/wiki/Secret_sharing

https://en.wikipedia.org/wiki/Rubber-hose_cryptanalysis

https://wiki.openssl.org/index.php/Enc
http://www.bashguru.com/2010/12/math-in-shell-scripts.html

https://github.com/openssl/openssl/issues/10145

Copyright