Skipjack and the World's First Encrypcurrency

Mr. Skipjack
Skipjack Currency British Virgin Island, 14 January 2018

Abstract. An **encrypcurrency** (or **encryp currency**) is a digital currency designed to work as a medium of exchange that uses encryption algorithm to secure its transactions, to control the creation of additional units, and to verify the transfer of currency. Encrypcurrency are classified as a subset of digital currencies based on underlying asset and are also classified as a subset of alternative currencies and virtual currencies. An encrypcurrency is a modern technology peer-to-peer a new version of electronic cash would allow online payments to be sent directly from one party to another with a centralized management and without going through a financial institution. The newly developed technology to speed up transaction processing by making the processing of communications between applications. The Superencryp Block platform design to be more efficient, which currently block chain have a problem on longer transaction had been the source of bottlenecks. This technology will implement in the Superencryp Block v0.0.1, it will increase transaction performance by approximately 100,000 times compared to the previous method in blockchain. With this technology, it has become possible to apply Superencryp Block technology to online transaction systems, which require high performance.

Motivation

The rise of virtual currency represents a radical transformation of how people store and transfer value. It took less than ten years for cryptocurrencies to become popular, compared to hundreds of years for paper money. Currently, only 8% of money globally exists in physical form; the vast majority of money is simply a digital balance, a ledger in a bank computer system. What people consider to be money is often simply an IOU, a virtual unit of account in a system over which they have little control. Within five years from now, most people will own a smartphone that can become their personal interface to the financial world, distributing financial control to a far greater degree than today. New payment systems are needed that are not controlled by a human central institution, and that provide better user experiences using mobile devices. Mainstream adoption of existing cryptocurrencies is held back by issues related to complexity, usability and scalability. In order for block chain and digital currency technologies to become mainstream and scale, new platforms was designed as" Superencryp Block" needed that are easier to use. Future digital currency protocols that seek to scale to billions of users must offer the stability of an asset like gold or USD within simple mobile interfaces.

SKIPJACK ENCRYPCURRENCY

Money, by its very nature, is controversial. It goes hand in hand with power, and for centuries has been integral to the operative status of the nation-state system. It can be used to control and to guide, to redeem and to oppress: like any tool in the hands of humans. At its essence, money is a ledger, a system used by society to keep score of who and what and when and where. Money is our way of recording distributed memory.

The evolution of money is therefore very important, and as technology impacts this part of the human experience like it has so many others, the principles and standards by which we engage technology and money together is of crucial consequence.

> ***"Skipjack was born from the weakness of crypto currency. Encrypcurrency breed from the algorithm technology which had more than 40 years hidden*** *from the public. Skipjack reborn to be the largest virtual currencies with 100 billion supply for next 100 years. Fiestel Core Network (FCN) was designed to be core engine in the skipjack central bank. It is forecast by year 2118, the world will be using skipjack with estimation USD 100 trillion market cap "*
> **Mr. SkipJack**
> **Inventor skipjack, Superencryp Block 2018-2118**

> *You heard about crypto currencies and digital currencies but do you know that the underlying of this technology was from the encryption algorithm. Skipjack is the new financial system and currency developed to revolutionize the payment system. Skipjack is the encryption technology lying for more than 30 years untouched and nobody is producing it. With the new protocol in place we are revolutionize the world with encrypcurrency. Welcome to the new world of currency. Despite of lack in security and anonymous field in bitcoin, skipjack is coming with* ***superencryp block cipher*** *technology using* ***Clipper chip*** *to address as new way of secured currency in a payment system.*
> *Mr. SkipJack, Inventor of Encrypcurrency*
> *January 2018*

History

Bitcoin, the most popular digital currency as of early 2008, has succeeded thus far due to its elegant and innovative design. Bitcoin was designed to be a p2p payment system, but has trended towards a platform resembling digital gold, with value that cannot be diluted by a single institution. Bitcoin is now held by millions of users, but often treated as a speculative investment rather than used as a medium-of-exchange.

Three issues with Bitcoin are the concentration of resources in the hands of a few, the large amount of energy consumed by the system, and its perception as a complex and unsafe platform. Bitcoin does create a digital asset in the form of scarce digital coins that have a shared perception of value. However, the emergence of large mining pools has led to uncertainty around long-term governance, and a small number of entities now control mining capacity. Future digital currencies should be more evenly distributed, broadly allocated and energy efficient.

Skipjack Preface

There are two kinds of cryptography in this world: cryptography that will stop your kid sister from reading your files, and cryptography that will stop major governments from reading your files.

If I take a letter, lock it in a safe, hide the safe somewhere in New York, then tell you to read the letter, that's not security. That's obscurity. On the other hand, if I take a letter and lock it in a safe, and then give you the safe along with the design specifications of the safe and a hundred identical safes with their combinations so that you and the world's best safecrackers can study the locking mechanism—and you still can't open the safe and read the letter—that's security.
For many years, this sort of cryptography was the exclusive domain of the military. The United States' National Security Agency (NSA), and its counterparts in the former Soviet Union, England, France, Israel, and elsewhere, have spent billions of dollars in the very serious game of securing their own communications while trying to break everyone else's. Private individuals, with far less expertise and budget, have been powerless to protect their own privacy against these governments.

During the last 20 years, public academic research in cryptography has exploded. While classical cryptography has been long used by ordinary citizens, computer cryptography was the exclusive domain of the world's militaries since World War II. Today, state–of–the–art computer cryptography is practiced outside the secured walls of the military agencies. The layperson can now employ security practices that can protect against the most powerful of adversaries—security that may protect against military agencies for years to come.

Do average people really need this kind of security? Yes. They may be planning a political campaign, discussing taxes, or having an illicit affair. They may be designing a new product, discussing a marketing strategy, or planning a hostile business takeover. Or they may be living in a country that does not respect the rights of privacy of its citizens. They may be doing something that they feel shouldn't be illegal, but for whatever reason, the data and communications are personal, private, and no one else's business.

In 1994, the Clinton administration approved the Escrowed Encryption Standard (including the Clipper chip and Fortezza card) and signed the Digital Telephony bill into law. Both of these initiatives try to ensure the government's ability to conduct electronic surveillance.
Some dangerously Orwellian assumptions are at work here: that the government has the right to listen to private communications, and that there is something wrong with a private citizen trying to keep a secret from the government. Law enforcement has always been able to conduct court–authorized surveillance if possible, but this is the first time that the people have been forced to take active measures to *make themselves available* for surveillance. These initiatives are not simply government proposals in some obscure area; they are preemptive and unilateral attempts to usurp powers that previously belonged to the people.

Clipper and Digital Telephony do not protect privacy; they force individuals to unconditionally trust that the government will respect their privacy. The same law enforcement authorities who illegally tapped Martin Luther King Jr.'s phones can easily tap a phone protected with Clipper. In the recent past, local police authorities have either been charged criminally or sued civilly in numerous jurisdictions—Maryland, Connecticut, Vermont, Georgia, Missouri, and Nevada—for conducting illegal wiretaps. It's a poor idea to deploy a technology that could some day facilitate a police state.

The lesson here is that it is insufficient to protect ourselves with laws; we need to protect ourselves with mathematics. Encryption is too important to be left solely to governments.

This encrypcurrency whitepaper gives us the overview on the development of Skipjack Financial protocol in the world of digital currency tools need to protect the financial own privacy; encrypcurrency products may be declared new to everyone, but the information will never be.

Let have a look in the other part of the digital world problem. What will we do when the world's data hits 163 Zettabytes in 2025? Zettabytes Now Needed to Describe Global Data Overload

In less than a decade from now, total worldwide data will swell to 163 zettabytes, which is equivalent to watching the entire Netflix catalog 489 million times. Most of this data will be created by enterprises and only about half will be be fully secured, says a Seagate study.

We — humanity, that is — created 4.4 zettabytes of data last year. This is expected to rise to 44 zettabytes by 2020. And no, I didn't make up the word "zettabytes." For scale, it is estimated that 42 zettabytes could store all human speech ever spoken. One zettabyte is around 250 billion DVDs — almost enough fit the whole *Friends* series.

All that data wouldn't be any use to us if we couldn't move it around quickly and reliably. We are constantly sending, receiving and streaming. We live in an age where anyone with a smart handheld device — and there are about 2.6 billion — can instantly become a video streamer. This large count doesn't even include big businesses and governments, most of which now do the majority of their communication digitally, adding to a prodigious amount of bits and bytes.

As connectivity continues to skyrocket, will we be able to move this data fast enough? Are we willing to pay the price to keep moving it faster and more reliably?

To c or Not To c

Current technology/wires are very fast, but our data creation and consumption is quickly catching up. Google Fiber, offering one of the fastest speeds available to regular consumers, has a transfer rate of one gigabit per second, and the Hibernia Express, a transatlantic fiber optic cable currently under construction for use in financial markets, will have a rate of 8.8 terabits per second (8800x faster than Google Fiber). So why not just go faster?

We are running up against the universal speed limit.

There are rules to the universe. Unfortunately, we don't know all of them. What we do know is that information has a speed limit: namely, c, the speed of light. Light travels about 300,000,000 meters per second. The two cables mentioned above, Google Fiber and the Hibernia Express, are both moving data at about 2/3 c (two-thirds the speed of light), limited by the refractive properties of the glass fiber. Basically, light bounces around and slows down inside the cable.

The Need for Speed

Outside of business, milliseconds may not mean millions of dollars, but today's consumer expectations are higher than ever. Viewers now consider instantaneous; on-demand access the norm. In this ecosystem, the word "buffering" means something has gone wrong. Slow load times for images while online shopping can mean clicking on the next site and never returning. And for the streaming service provider or the online retailer, this means dents in the bottom line.

Are we willing to pay the price to keep moving data faster and more reliably?

The rate at which we create data is not going to slow down anytime soon. The advent of the Internet of Things looms on the horizon — a predicted 25 billion devices (three per person on Earth) each producing, sending and receiving data. Is our infrastructure ready for that cascade of data?

The Age of The Plateau

Notice how the line on the infographic below flattens out over the last few decades? We've made huge advances in previous decades, but now the rate of progress is affected by very different struggles. The most advanced technologies are only a few percentage points away from the speed of light — but the technology to reliably utilize that speed may be many years and many millions of dollars away.

An increase of a small percentage in speed requires intensive resources. The Hibernia Express will shave off 6 milliseconds of latency for transatlantic transmission. It is estimated that the project will cost more than $300 million, and firms will pay millions for access to the line. The secret behind the Hibernia's speed is quite simple: They laid the cable in a straighter line across the Atlantic.

For the average consumer, and even those with intense speed needs, like financial markets, the faster transfer doesn't make that much of a difference. But when we consider the sheer volume of data that will need to be moved constantly, reliably and without issue, the importance of these incremental steps toward better infrastructure is clear. Every tenth of a percentage point will be hugely impactful.

Faster, Better, Stronger

Luckily, breakthroughs are coming quickly, albeit often at great price. Researchers in the U.K. have created fiber cables that move data at 99.73 percent the speed light, and combined them with technologies that allow for incredible 73.7 terabits per second — that's a 5GB HD movie downloaded in 1 second. These cables, however, are so far only efficient for very short distances, so further work is needed.

A good way to avoid losing speed to the physical media you are sending the light through is to avoid it altogether. Microwave relays, which are sent through (mostly) empty air, are able to travel much faster. Relays must be built as a series of stationary towers, so the viability of crossing an ocean in such a manner would make for some serious construction challenges. Some have proposed a series of stationary barges holding towers and, in a truly sci-fi twist, a network of hovering drones that would transfer the signal over short distances using lasers.

All this progress will come at great cost. The financial sector has typically led the charge because of their reliance on absolute top speeds, and those technologies, proven in the fire of the international markets, eventually make their way into general business use.

We are spending billions on this development; progress is slow compared to the increase in speeds in previous decades. Every fraction of the speed of light matters if we expect to keep up with the incredible amount of data we will be capable of producing in the coming years.

Milliseconds don't matter until there are millions of them, and millions upon millions of people expecting to consume at speeds close to the speed of light.

In this whitepaper we discussed the new invention of the law of digital encryption and it is name as Jack's Law.

Jack's *law* *is the observation that the number of network computing in a dense of data doubles about every two years. The observation is named after Mr. Skipjack, the scientist of Skipjack Corporation, who's in the Encrypcurrency paper described a doubling every year in the number of network connected per data network, and projected this rate of growth would continue for at least another decade. The period is often quoted as 18 months because of we predicted that network performance and efficiency protocols would double every 18 months (being a combination of the effect of more data and the network connectivity be more efficient and faster based on protocols and hardware improvement). This will be a problem solver of the increase of internet traffic that forecasts surge in IP traffic driven by number of factors, including growth of the Internet of Things.*

More Internet users, more connected devices, faster broadband speeds and more video all add up to one thing: a heck of a lot of IP traffic. Latest Visual Networking Index (VNI) shows global Internet traffic growing at an eye-popping rate, reaching 2 zettabytes per year by 2019.

Two zettabytes is 12 times more than all the IP traffic generated in 2009.While it took 32 years -- from 1984 to 2016 -- to reach the first zettabyte mark, it will take only three more years to reach 2 zettabytes.

Global IP traffic will grow at a compound annual growth rate of 23% between 2014 and 2019, reaching 168 exabytes per month in four years VNI forecast.

Jack's prediction will be use in the future internet industry to guide long-term planning and to set targets for research and development. Advancements in digital economy are strongly linked to Jack's law: Jack's law describes that the connectivity, the storage and the speed shall improve from the change of split protocol in the algorithm in the encryption of data technology. It is a factor to driving force of technological and social change, productivity, and economic growth.

Jack's law is an observation and projection of a trend and not a physical or natural law. In general, it is not logically sound to extrapolate from the historical growth rate into the indefinite future.

When Jack's law applied in financial technology it will increase the store value of currency based on demand and technology improvement in protocols efficiency design and connectivity solving the issue of speed of data, storage and connectivity. Superencryp block it will increase the efficiency as centralized ledger in financial transaction to transfer the virtual currency more efficient where to the speed more than 1 million transactions per second.

What is the theory behind Skipjack?

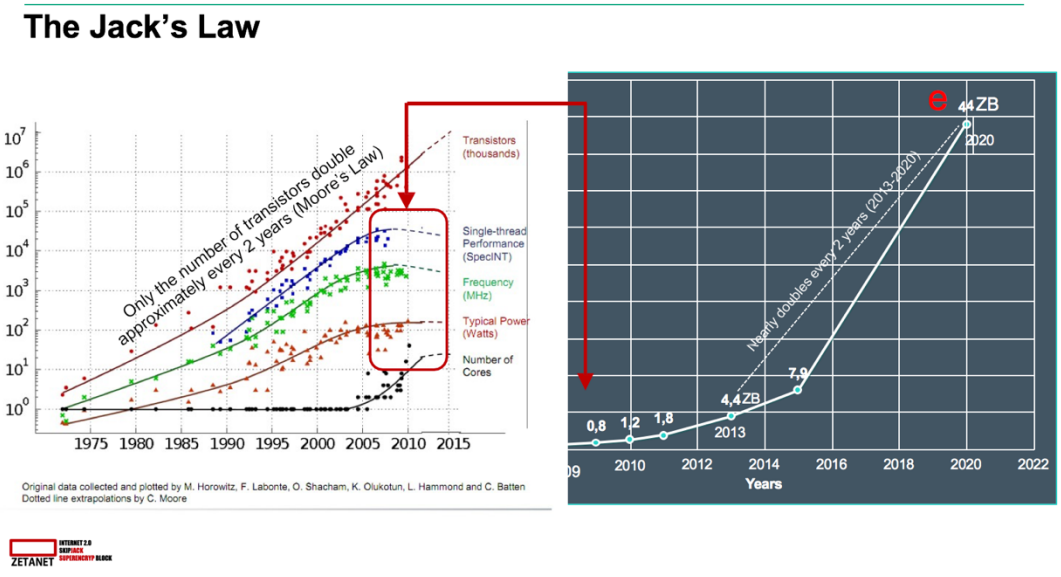
Skipjack encrypcurrency is the virtual money that was birth based on the Jack's Law. It was designed to use under principle of Jack's law as Superencryp Block with Key Exchange Algorithm.

A **superencrypblock**, originally **superencryp block**, is a centralized ledger called *superencryp*, which are linked using cryptography and encryption technology. Superencrypblock which are readable by the public are widely used by encrypcurrency.

Each superencryp contains a cryptographic hash and encryption key exchange algorithm of the previous superencryp, double HMAC, a timestamp, and transaction data. By design, a superencrypblock is resistant to modification of the data. It is "a centralized ledger controlled by a system called ***Fiestel Core Network (FCN)*** that can record transactions between two parties efficiently and in a verifiable and permanent way. For use as a centralized ledger a superencrypblock is typically managed by a peer-to-peer network collectively adhering to a protocol for inter-node communication and validating new superencryp by FCN. Once recorded, the data in any given superencryp cannot be altered retroactively without alteration of all subsequent superencryp, which requires consensus of the network majority and FCN.

Though superencrypblock records are not unalterable, superencryp may be considered secure by design and exemplify a centralized computing system with high double HMAC protocol verification method.

Superencrypblock was invented by Mr. Skipjack in 2018 to serve as the public transaction ledger of the encrypcurrency skipjack dime. The invention of the superencryp block made it the first digital currency to develop highly encrypted application of new internet including to solve the transaction speed, security on the central server. The skipjack design has inspired other applications such as the Internet 2.0 in the Skipjack Zetanet project.



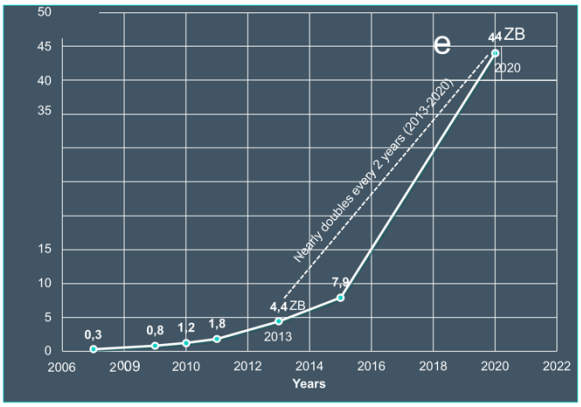
Jack's Law

Jack's law is the observation that the number of network computing in a dense of data doubles about every two years. The observation is named after Mr. Skipjack, the scientist of Skipjack Corporation, whose in the Encrypcurrency paper described a doubling every year in the number of network connected per data network, and projected this rate of growth would continue for at least another decade. The period is often quoted as 18 months because of we predicted that network performance and efficiency protocols would double every 18 months (being a combination of the effect of more data and the network connectivity be more efficient and faster based on protocols and hardware improvement). This will be a problem solver of the increase of internet traffic that forecasts surge in IP traffic driven by number of factors, including growth of the Internet of Things.

The Jack's Law

The invention states the data in network computing nodes is increasing nearly doubles every year and a split protocol is solving the network by increasing 10 times algorithm efficiency within the communication network.

Mr. Skipjack, Chief Scientist



Based on the fundamentals of Jack's law, the superencrypblock is developed to the application for financial technology and the next Internet 2.0 as we described here as Zetanet. Zetanet is described as a new internet protocols to be released and demonstrate in year 2020 for the application of the next Internet 2.0.

Under Zetanet project by Skipjack Corporation it will be used to solve the problem when the world's data hits 163 Zettabytes in 2025. Zettabytes now needed to describe global data overload and will be solved by the new Jack's law theorem.

Encrypcurrency protocols

Skipjack Protocol Building Blocks

Introduction to Protocols

The whole point of encrypcurrency is to solve financial transactions problems. (Actually, that's the whole point of computers—something many people tend to forget.) Encrypcurrency solves problems that involve secrecy, authentication, integrity, and dishonest people when do the transactions.

A **protocol** is a series of steps, involving two or more parties, designed to accomplish a task. This is an important definition. A "series of steps" means that the protocol has a sequence, from start to finish. Every step must be executed in turn, and no step can be taken before the previous step is finished. "Involving two or more parties" means that at least two people are required to complete the protocol; one person alone does not make a protocol. A person alone can perform a series of steps to accomplish a task (like baking a cake), but this is not a protocol. (Someone else must eat the cake to make it a protocol.) Finally, "designed to accomplish a task" means that the protocol must achieve something. Something that looks like a protocol but does not accomplish a task is not a protocol—it's a waste of time.

Protocols have other characteristics as well:

- Everyone involved in the protocol must know the protocol and all of the steps to follow in advance.
- Everyone involved in the protocol must agree to follow it.
- The protocol must be unambiguous; each step must be well defined and there must be no chance of a misunderstanding
- The protocol must be complete; there must be a specified action for every possible situation.

The protocols in this encrypcurrency design are organized as a series of steps. Execution of the protocol proceeds linearly through the steps, unless there are instructions to branch to another step. Each step involves at least one of two things: computations by one or more of the parties, or messages sent among the parties.

An **encrypcurrency protocol** is a protocol that uses cryptography. The parties can be friends and trust each other implicitly or they can be adversaries and not trust one another to give the correct time of day. An encrypcurrency protocol involves some cryptographic algorithm, but generally the goal of the protocol is something beyond simple secrecy. The parties participating in the protocol might want to share parts of their secrets to compute a value, jointly generate a random sequence, convince one another of their identity, or simultaneously sign a contract. The whole point of using cryptography in a protocol is to prevent or detect eavesdropping and cheating. If we have never seen these protocols before, they will radically change our ideas of what mutually distrustful parties can accomplish over a computer network. In general, this can be stated as:

- It should not be possible to do more or learn more than what is specified in the protocol.

This is a lot harder than it looks. In some of them it is possible for one of the participants to cheat the other. In others, it is possible for an eavesdropper to subvert the protocol or learn secret information. **Some protocols fail because the designers weren't thorough enough in their requirements definitions. Others fail because their designers weren't thorough enough in their analysis. Like algorithms, it is much easier to prove insecurity than it is to prove security.**

The Skipjack Encrypcurrency Protocol

Skipjack mission is to create a verified and centralized distributed platform for value exchange, a global currency protocol that enables a payment system that is easier, safer, and faster to use than paper money or cryptocurrency. In addition, Skipjack will ensure that the majority of the economic value generated by the platform is fairly distributed to the community through accounts created, to create a more balanced distribution of resources.

Skipjack seeks to address three issues within digital currencies: verification of speed transactions, usability of applications, and efficiency of transactions. Skipjack makes significant improvements by *1) forming a matching block superencryp to verify network 2) increasing overall skipjack dime supply, and creating multiple efficiency platform and mobile encryptrade apps, and 3) designing a centralized system managed by Fiestel Core Network (FCN) as central bank with less human interface. 4) Improving security breach by designing a hybrid protocol and multi layer encryption.*

Skipjack is a new currency that the new term as **encrypcurrency**. Skipjack is a general purpose encrypt digital currency. It is **programmable money backed with asset**. An **encrypcurrency** (or **encryp currency**) is a digital asset designed to work as a medium of exchange that uses encryption algorithm to secure its transactions, to control the creation of additional units, and to verify the transfer of assets. Encrypcurrency are classified as a subset of digital currencies and are also classified as a subset of alternative currencies and virtual currencies.

Skipjack is an encryption algorithm for the transmission of information (voice data in particular). It uses the Diffie-Hellman key exchange algorithm for the distribution of the cryptographic session keys between peers. The Skipjack algorithm was classified as an NSA Type 2 encryption product. It is using Clipper Chip for encrypt transaction. The algorithm was initially classified as SECRET, so that it could not be examined in the usual manner by the encryption research community. It used a **512-bit key** and a symmetric cipher algorithm, similar to DES. Data is encrypted in blocks of 64 bits, using an unbalanced Feistel network with 32 rounds with more security than bitcoin and difficult to brute force. The new skipjack algorithm was developed to overcome the existing traditional digital currency which currently having a lot of weakness.

Skipjack, created in 2018, is the **first centralized encrypcurrency**. Skipjack and its derivatives use **Feistel Core Network (FCN) designed with Artificial Intelligence (AI) Superencryp Block** act as Central Bank that centralized control as same to centralized electronic money and central banking systems. The centralized control is related to the use of skipjack's **superencryp blockchiper** transaction database in the role of a distributed ledger.

Encryp currency (encryp money) or called as Skipjack Dime is a type of currency available only in digital form, not in physical (such as banknotes and coins). It exhibits properties similar to physical currencies, but allows for instantaneous transactions and borderless transfer-of-ownership. Examples include virtual currencies and cryptocurrencies or even central bank issued "digital base money". Like traditional money, these currencies may be used to buy physical goods and services, but may also be restricted to certain communities such as for use inside an on-line game or social network. Encryp currency is a money balance recorded electronically on a stored-value card or other device. Another form of electronic money is network money, allowing the transfer of value on computer networks, particularly the Internet. Encryp money is also a claim on a private bank or other financial institution such as bank deposits.

Understanding the Skipjack Protocol Design: The Players

To help demonstrate understand how skipjack protocols, I have enlisted the aid of several people (see Table 1.1). Alice and Bob are the first two. They will perform all general two-person protocols. As a rule, Alice will initiate all protocols and Bob will respond. If the protocol requires a third or fourth person, Carol and Dave will perform those roles. Other actors will play specialized supporting roles; they will be introduced later.

Arbitrated Protocols

An **arbitrator** is a disinterested third party trusted to complete a protocol. Disinterested means that the arbitrator has no vested interest in the protocol and no particular allegiance to any of the parties involved. Trusted means that all people involved in the protocol accept what he says as true, what he does as correct, and that he will complete his part of the protocol. Arbitrators can help complete protocols between two mutually distrustful parties.

In the real world, lawyers are often used as arbitrators. For example, Alice is selling a car to Bob, a stranger. Bob wants to pay by check, but Alice has no way of knowing if the check is good. Alice wants the check to clear before she turns the title over to Bob. Bob, who doesn't trust Alice any more than she trusts him, doesn't want to hand over a check without receiving a title.

TABLE 1.1
Dramatis Personae

Alice	First participant in all the protocols
Bob	Second participant in all the protocols
Carol	Participant in the three- and four-party protocols
Dave	Participant in the four-party protocols
Eve	Eavesdropper
Mallory	Malicious active attacker
FCN	Trusted arbitrator
Walter	Warden; he'll be guarding Alice and Bob in some protocols
Peggy	Prover
Victor	Verifier

Skipjack Basic Protocols

Encryption Algorithms Contained in Skipjack

Skipjack may contain one or more of the following encryption algorithms.
Accordingly, Skipjack containing encryption are subject to the U.S. Export Administration Regulations and may be controlled for National Security (NS), Anti Terrorism (AT), and/or Encryption (EI) reasons.

The primary function of encryption in Skipjack falls into two categories:

1.Data Privacy (confidentiality - encrypting user data)

2.Administrative - OAM&P (secure network management - administrative network functions)

Data Hashing/Authentication
MD4
MD5 SHA-1 RIPEMD160 MICHAEL HMAC MMH CRC32

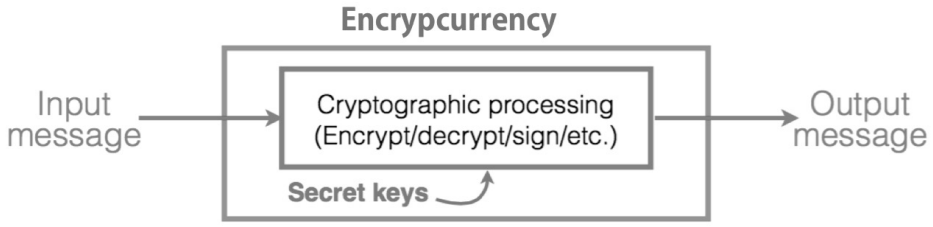
Key Exchange/ Asymmetric Algorithm Strength
DSA 1024, 1024/1536
Elliptic Curve Diffie-Hellman (ECDH) 163
RSA 1024, 2048, 4096
DSA 1024,2048 Diffie Hellman 1024, 1536 ElGamal 384

Data Security Encryption
Blowfish 128 CAST 128 DES-56 56, 64 Triple DES 112, 168, 192 DESX 56/64 RC2 40, 64, 128 RC4—128 40, 128 RC5 128 AES-128/l92/256 ARCFOUR 128 SEAL 160 IDEA 128 Skipjack

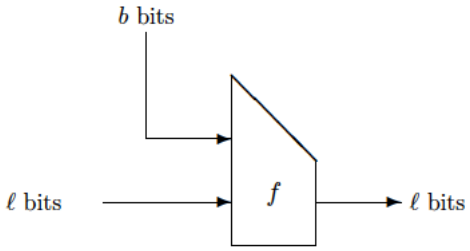
Message Authentication Codes

A **message authentication code** (MAC), also known as a data authentication code (DAC) use in Skipjack protocol, is a one-way hash function with the addition of a secret key. The hash value is a function of both the pre-image and the key. The theory is exactly the same as hash functions, except only someone with the key can verify the hash value. Skipjack can create a MAC out of a hash function or a block encryption algorithm; there are also dedicated MACs.

A message authentication code, or MAC, is a key-dependent one-way hash function. MACs have the same properties as the one-way hash functions, but they also include a key. Only someone with the identical key can verify the hash. They are very useful to provide authenticity without secrecy. MACs can be used to authenticate files between users. They can also be used by a single user to determine if his files have been altered, perhaps by a virus. A user could compute the MAC of his files and store that value in a table. If the user used instead a one-way hash function, then the virus could compute the new hash value after infection and replace the table entry. A virus could not do that with a MAC, because the virus does not know the key.



A common encrypcurrency technique is to encrypt each individual transaction with a separate key. This is called a session key, because it is used for only one particular communications session. Session keys are useful because they only exist for the duration of the communication.



A compression function. ($b = 512$, $\ell = 128$ in MD5.).

Key Exchange with Symmetric Cryptography

This protocol assumes that Alice and Bob, users on a network, each share a secret key with the Key Distribution Center (KDC) —FCN in our protocols. These keys must be in place before the start of the protocol. (The protocol ignores the very real problem of how to distribute these secret keys; just assume they are in place and Mallory has no idea what they are.)

(1) Alice calls FCN and requests a session key to communicate with Bob.

(2) FCN generates a random session key. He encrypts two copies of it: one in Alice's key and the other in Bob's key. FCN sends both copies to Alice.

(3) Alice decrypts her copy of the session key.

(4) Alice sends Bob his copy of the session key.

(5) Bob decrypts his copy of the session key.

(6) Both Alice and Bob use this session key to communicate securely.

This protocol relies on the absolute security of FCN, who is more likely to be a trusted computer program than a trusted individual. If Mallory corrupts FCN, the whole network is compromised. He has all of the secret keys that FCN shares with each of the users; he can read all past communications traffic that he has saved, and all future communications traffic. All he has to do is to tap the communications lines and listen to the encrypted message traffic.
The other problem with this system is that FCN is a potential bottleneck. He has to be involved in every key exchange. If FCN fails, that disrupts the entire system.

Key Exchange with Public-Key Cryptography

The basic hybrid encrypsystem was discussed. Alice and Bob use public-key cryptography to agree on a session key, and use that session key to encrypt data. In some practical implementations, both Alice's and Bob's signed public keys will be available on a database. This makes the key-exchange protocol even easier, and Alice can send a secure transaction to Bob even if he has never heard of her:

(1) Alice gets Bob's public key from the KDC.

(2) Alice generates a random session key, encrypts it using Bob's public key, and sends it to Bob.

(3) Bob then decrypts Alice's message using his private key.

Both of them encrypt their transactions using the same session

Man-in-the-Middle Attack

While Eve cannot do better than try to break the public-key algorithm or attempt a ciphertext-only attack on the ciphertext, Mallory is a lot more powerful than Eve. Not only can he listen to messages between Alice and Bob, he can also modify messages, delete messages, and generate totally new ones. Mallory can imitate Bob when talking to Alice and imitate Alice when talking to Bob.

Here's how the attack works:

(1) Alice sends Bob her public key. Mallory intercepts this key and sends Bob his own public key.

(2) Bob sends Alice his public key. Mallory intercepts this key and sends Alice his own public key.

(3) When Alice sends a message to Bob, encrypted in "Bob's" public key, Mallory intercepts it. Since the message is really encrypted with his own public key, he decrypts it with his private key, re-encrypts it with Bob's public key, and sends it on to Bob.

(4) When Bob sends a message to Alice, encrypted in "Alice's" public key, Mallory intercepts it. Since the message is really encrypted with his own public key, he decrypts it with his private key, re-encrypts it with Alice's public key, and sends it on to Alice.

Even if Alice's and Bob's public keys are stored on a database, this attack will work. Mallory can intercept Alice's database inquiry and substitute his own public key for Bob's. He can do the same to Bob and substitute his own public key for Alice's. Or better yet, he can break into the database surreptitiously and substitute his key for both Alice's and Bob's. Then he simply waits for Alice and Bob to talk with each other, intercepts and modifies the messages, and he has succeeded.

This **man-in-the-middle attack** works because Alice and Bob **have no way to verify** that they are talking to each other. Assuming Mallory doesn't cause any noticeable network delays, the two of them have no idea that someone sitting between them is reading all of their supposedly secret communications.

Skipjack Interlock Protocol

The skipjack design applies the **interlock protocol**, has a **good chance of foiling the man-in-the-middle attack.** Here's how it works:

(1) Alice sends Bob her public key.

(2) Bob sends Alice his public key.

(3) Alice encrypts her message using Bob's public key. She sends half of the encrypted message to Bob.

(4) Bob encrypts his message using Alice's public key. He sends half of the encrypted message to Alice.

(5) Alice sends the other half of her encrypted message to Bob.

(6) Bob puts the two halves of Alice's message together and decrypts it with his private key. Bob sends the other half of his encrypted message to Alice.

(7) Alice puts the two halves of Bob's message together and decrypts it with her private key.

The important point is that half of the message is useless without the other half; it can't be decrypted. Bob cannot read any part of Alice's message until step (6); Alice cannot read any part of Bob's message until step (7). There are a number of ways to do this:

— If the encryption algorithm is a block algorithm, half of each block (e.g., every other bit) could be sent in each half message.

— Decryption of the message could be dependent on an initialization vector which could be sent with the second half of the message.

— The first half of the message could be a one-way hash function of the encrypted message and the encrypted message itself could be the second half.

To see how this causes a problem for Mallory, let's review his attempt to subvert the protocol. He can still substitute his own public keys for Alice's and Bob's in steps (1) and (2). But now, when he intercepts half of Alice's message in step (3), **he cannot decrypt it with his private key and re-encrypt it with Bob's public key.** He must invent a totally new message and send half of it to Bob. When he intercepts half of Bob's message to Alice in step (4), he has the same problem. He cannot decrypt it with his private key and re-encrypt it with Alice's public key. He has to invent a totally new message and send half of it to Alice. By the time he intercepts the second halves of the real messages in steps (5) and (6), it is too late for him to change the new messages he invented. The transactions between Alice and Bob will necessarily be completely different.

"Mallory could possibly get away with this scheme. If he knows Alice and Bob well enough to mimic both sides of a conversation between them, they might never realize that they are being duped. But surely this is much harder than sitting between the two of them, intercepting and reading their messages. But he can't!".

Key Exchange with Digital Signatures

Implementing skipjack digital signatures during a session-key exchange protocol circumvents this man-in-the-middle attack as well.

FCN signs both Alice's and Bob's public keys. The signed keys include a signed certification of ownership. When Alice and Bob receive the keys, they each verify FCN's signature. Now they know that the public key belongs to that other person. The key exchange protocol can then proceed.
Mallory has serious problems. He cannot impersonate either Alice or Bob because he doesn't know either of their private keys. He cannot substitute his public key for either of theirs because, while he has one signed by FCN, it is signed as being Mallory's. All he can do is listen to the encrypted traffic go back and forth or disrupt the lines of communication and prevent Alice and Bob from talking.

This protocol uses FCN, but the risk of compromising the KDC is less than the first protocol. If Mallory compromises FCN (breaks into the KDC), all he gets is FCN's private key. This key enables him only to sign new keys; it does not let him decrypt any session keys or read any message traffic. To read the traffic, Mallory has to impersonate a user on the network and trick legitimate users into encrypting messages with his phony public key.

Mallory can launch that kind of attack. With FCN's private key, he can create phony signed keys to fool both Alice and Bob. Then, he can either exchange them in the database for real signed keys, or he can intercept users' database requests and reply with his phony keys. This enables him to launch a man-in-the-middle attack and read people's communications.

This attack will work, but remember that Mallory has to be able to intercept and modify messages. In some networks this is a lot more difficult than passively sitting on a network reading messages as they go by.

On a broadcast channel, such as a radio network, it is almost impossible to replace one message with another—although the entire network can be jammed. On computer networks this is easier and seems to be getting easier every day. Consider IP spoofing, router attacks, and so forth; active attacks don't necessarily mean someone down a manhole with a datascope, and they are not limited to three-letter agencies.

Skipjack Key and Message Transmission

Alice and Bob need not complete the key-exchange protocol before exchanging transactions. In this protocol, Alice sends Bob the transaction, *M*, without any previous key exchange protocol:

(1) Alice generates a random session key, *K*, and encrypts *M* using *K*.

$$E_K(M)$$

(2) Alice gets Bob's public key from the database.

(3) Alice encrypts *K* with Bob's public key.

$$E_B(K)$$

(4) Alice sends both the encrypted transaction and encrypted session key to Bob.

$$E_K(M),\ E_B(K)$$

For added security against man-in-the-middle attacks, Alice can sign the transmission.

(5) Bob decrypts Alice's session key, *K*, using his private key.

(6) Bob decrypts Alice's message using the session key.

This hybrid system is how public-key encrypcurrency is used in a communications and transactions system. It can be combined with digital signatures, timestamps, and any other security protocols.

Skipjack Key and Message Broadcast

There is no reason Alice can't send the encrypted message to several people. In this example, Alice will send the encrypted message to Bob, Carol, and Dave:

(1) Alice generates a random session key, *K*, and encrypts *M* using *K*.

$$E_K(M)$$

(2) Alice gets Bob's, Carol's, and Dave's public keys from the database.

(3) Alice encrypts *K* with Bob's public key, encrypts *K* with Carol's public key, and then encrypts *K* with Dave's public key.

$$E_B(K),\ E_C(K),\ E_D(K)$$

(4) Alice broadcasts the encrypted message and all the encrypted keys to anybody who cares to receive it.

$$E_B(K),\ E_C(K),\ E_D(K),\ E_K(M)$$

(5) Only Bob, Carol, and Dave can decrypt the key, *K*, each using his or her private key.

(6) Only Bob, Carol, and Dave can decrypt Alice's message using *K*.

This protocol can be implemented on a store-and-forward network. A central server can forward Alice's message to Bob, Carol, and Dave along with their particular encrypted key. The server doesn't have to be secure or trusted, since it will not be able to decrypt any of the messages

Keyed-Hash Message Authentication Codes (HMAC)

Message authentication codes (MACs) provide data authentication and integrity protection. Two types of algorithms for computing a MAC have been approved: 1) MAC algorithms that are based on approved block cipher algorithms and 2) MAC algorithms that are based on hash functions, called HMAC algorithms, that are specified in FIPS 198-1. An output from an HMAC algorithm is called an HMAC output. The HMAC output is either used in its entirety, or is truncated, when it is transmitted for subsequent verification. The transmitted value is called a MacTag. The HMAC **algorithm requires the use of a secret key** that is shared between the entity that generates the HMAC output (e.g., a message sender), and the entity (or entities) that need to verify the transmitted MacTag (message receiver(s)). The HMAC output is generated from a secret key and the string of "text" to be MACed (e.g., a message to be sent) using the HMAC algorithm. The MacTag is provided to the MacTag verifier, along with the "text" that was MACed (e.g., the sender transmits both the MacTag and the message to the intended receiver). The verifier computes an HMAC output on the received "text" using the same key and HMAC algorithm that were (purportedly) used to generate the received MacTag, generates a (new) MacTag (either a full or truncated HMAC output), and then compares the verifier-generated MacTag with the received MacTag. If the two values match, the "text" has been correctly received and the verifier is assured that the entity that generated the MacTag is a member of the community of users that share the key. The security strength provided by the HMAC algorithm depends on the security strength of the **HMAC key, the underlying hash algorithm and the length of the MacTag.**

The HMAC Key

The security strength of the HMAC algorithm depends, in part, on the security strength of the HMAC key, K. A HMAC key shall have a security strength that meets or exceeds the security strength required to protect the data over which the HMAC is computed. The HMAC key shall be kept secret. When the secrecy of the HMAC key, K, is not preserved, an adversary that knows K, may impersonate any of the users that share that key in order to generate MacTags that seem to be authentic (i.e., MacTags that can be verified and are subsequently presumed to be authentic). HMAC keys shall be generated as specified in SP 800-133 [SP 800-133].

Truncation of HMAC Output

When an application truncates the HMAC output to generate a MacTag to a desired length, λ, the λ left-most bits of the HMAC output shall be used as the MacTag. However, the output length, λ, shall be no less than 32 bits. For example, a low bandwidth channel or a desired high efficiency computation application such as audio or video casting application might use 32-bit MacTags.

Security Effect of the HMAC Key

Let C denote the bit length of the internal hash value that is denoted H in FIPS 180-4. (This H is often called the "chaining value" in descriptions of Merkle–Damgård-style hash functions.) Note that C is not (necessarily) equal to L, the bit length of the hash function's output (see, for example, SHA-384 or SHA-512/t for any $t < 512$, for which $L< 512 = C$). In all currently approved hash functions, $L \leq C$ (with $L = C$ for SHA-1, SHA-256, and SHA-512). The effective security strength of the HMAC key is the minimum of the security strength of K and the value of 2C. That is, security strength = min(security strength of K, 2C). For example, if the security strength of K is 128 bits, and

SHA-1 is used, then the effective security strength of the HMAC key is 128 bits, since for SHA-1, 2C= 320. Note that, in this example, even if the security strength of K is greater than 320 bits, the effective security strength of the key is limited to 320 bits (the value of 2C for SHA-1). In general, there is no benefit in generating K with more than 2C bits of security. In particular, it is not sensible to generate K with a bit length that exceeds the input block size of the approved L-bit hash function employed in the HMAC construction. Such a K is hashed, and the resulting L-bit value is used instead. Returning to the SHA-1 example, suppose that the key K has a bit length greater than 512 (the input block size for SHA-1). Instead of using K directly, HMAC replaces K by its 160-bit SHA-1 hash value, and so the effective security strength of that choice of K is no more than 160 bits.

Security of the HMAC Values

The successful verification of a MacTag does not completely guarantee that the accompanying text is authentic; there is a slight chance that an adversary with no knowledge of the HMAC key, K, can present a (MacTag, text) pair that will pass the verification procedure. From the perspective of an adversary that does not know the HMAC key K (i.e., the adversary is not among the community of users that share the key), the assurance of authenticity provided by a MacTag depends on its length and on the number of failed MacTag verifications allowed by a system for each value of the HMAC key. Let 2^t be the number of failed MacTag verifications allowed by a system for each value of the HMAC key. The MacTag length, λ, and the value of t need to be chosen to avoid an unacceptable probability of falsely accepting forged data. The likelihood of accepting forged data as authentic is (1/2)(λ - t). For example, if λ is 32, and a system allows 212 failed MacTag verifications for any given value of the HMAC key, then the likelihood of accepting forged data is (1/2)20. In order to increase assurance of authenticity, either λ would need to be increased, or the number of allowed failed MacTag verifications for each value of the HMAC key would need to be decreased. To avoid having an unacceptable probability of falsely accepting forged data at any time, the value of the HMAC key must be changed to a new value before the number of failed MacTag authentications reaches the maximum allowed number (2^t). For the example above, with λ = 32 and t = 20, the HMAC key must be changed before the number of failed MacTag verifications reaches 212.

The table below provides the likelihoods of accepting forged data for different MacTag lengths and allowed numbers of MAC verifications using a given value of the HMAC key. This table is intended to assist the implementers of HMAC applications in security sensitive systems to assess the security risk associated with using MacTags.

λ	Number of Failed Verifications Allowed (2^t): 2^{20}	2^{30}	2^{35}
40	2^{-20}	2^{-10}	2^{-5}
64	2^{-44}	2^{-34}	2^{-29}
96	2^{-76}	2^{-66}	2^{-61}

Each 2^{-x} entry displayed above is the probability of accepting forged data given the indicated choices for the MacTag length and the number of failed MacTag verifications allowed for a fixed value of the HMAC key. If the probability is not acceptable for the system, the HMAC key shall be changed to a new value before the number of failed MAC verifications reaches 2^t. A commonly acceptable length for the MacTag is 64 bits; MacTags with lengths shorter than 64 bits are discouraged.

HMAC Authentication – Cryptographic signature and verification of messages.

The hmac module implements keyed-hashing for message authentication, as described in **RFC 2104**

The HMAC algorithm can be used to verify the integrity of information passed between applications or stored in a potentially vulnerable location. The basic idea is to generate a cryptographic hash of the actual data combined with a shared secret key. The resulting hash can then be used to check the

Example

Creating the hash is not complex. Here's a simple example which uses the default MD5 hash algorithm:

```
import hmac

digest_maker = hmac.new('secret-shared-key-goes-here')

f = open('lorem.txt', 'rb')
try:
    while True:
        block = f.read(1024)
        if not block:
            break
        digest_maker.update(block)
finally:
    f.close()

digest = digest_maker.hexdigest()
print digest
```

When run, the code reads its source file and computes an HMAC signature for it:

```
$ python hmac_simple.py

4bcb287e284f8c21e87e14ba2dc40b16
```

SHA vs. MD5

Although the default cryptographic algorithm for hmac isMD5, that is not the most secure method to use. MD5 hashes have some weaknesses, such as collisions (where two different messages produce the same hash). The SHA-1 algorithm is considered to be stronger, and should be used instead.

```
import hmac
import hashlib

digest_maker = hmac.new('secret-shared-key-goes-here', '', hashlib.sha1)

f = open('hmac_sha.py', 'rb')
try:
    while True:
```

```
        block = f.read(1024)
        if not block:
            break
        digest_maker.update(block)
finally:
    f.close()

digest = digest_maker.hexdigest()
print digest
```

hmac.new() takes 3 arguments. The first is the secret key, which should be shared between the two endpoints which are communicating so both ends can use the same value. The second value is an initial message. If the message content that needs to be authenticated is small, such as a timestamp or HTTP POST, the entire body of the message can be passed to new() instead of using the update() method. The last argument is the digest module to be used. The default is hashlib.md5. The previous example substitutes hashlib.sha1.

```
$ python hmac_sha.py

69b26d1731a0a5f0fc7a92fc6c540823ec210759
```

Binary Digests

The first few examples used the hexdigest() method to produce printable digests. The hexdigest is is a different representation of the value calculated by the digest() method, which is a binary value that may include unprintable or non-ASCII characters, including NULs. Some web services (Google checkout, Amazon S3) use the base64 encoded version of the binary digest instead of the hexdigest.

```
import base64
import hmac
import hashlib

f = open('lorem.txt', 'rb')
try:
    body = f.read()
finally:
    f.close()


digest = hmac.new('secret-shared-key-goes-here', body, hashlib.sha1).digest()
print base64.encodestring(digest)
```

The base64 encoded string ends in a newline, which frequently needs to be stripped off when embedding the string in HTTP headers or other formatting-sensitive contexts.

```
$ python hmac_base64.py

olW2DoXHGJEKGU0aE9fOwSVE/o4=
```

Applications

HMAC authentication should be used for any public network service, and any time data is stored where security is important. For example, when sending data through a pipe or socket, that data should be signed and then the signature should be tested before the data is used. The extended example below is available in the hmac_pickle.py file as part of the PyMOTW source package.

First, let's establish a function to calculate a digest for a string, and a simple class to be instantiated and passed through a communication channel.

```
import hashlib
import hmac
try:
    import cPickle as pickle
except:
    import pickle
import pprint
from StringIO import StringIO


def make_digest(message):
    "Return a digest for the message."
    return hmac.new('secret-shared-key-goes-here', message, hashlib.sha1).hexdigest()


class SimpleObject(object):
    "A very simple class to demonstrate checking digests before unpickling."
    def __init__(self, name):
        self.name = name
    def __str__(self):
        return self.name
```

Next, create a StringIO buffer to represent the socket or pipe. We will using a naive, but easy to parse, format for the data stream. The digest and length of the data are written, followed by a new line. The serialized representation of the object, generated by picklefollows. In a real system, we would not want to depend on a length value, since if the digest is wrong the length is probably wrong as well. Some sort of terminator sequence not likely to appear in the real data would be more appropriate.

For this example, we will write two objects to the stream. The first is written using the correct digest value.

```
# Simulate a writable socket or pipe with StringIO
out_s = StringIO()

# Write a valid object to the stream:
#  digest\nlength\npickle
o = SimpleObject('digest matches')
pickled_data = pickle.dumps(o)
digest = make_digest(pickled_data)
header = '%s %s' % (digest, len(pickled_data))
```

```
print '\nWRITING:', header
out_s.write(header + '\n')
out_s.write(pickled_data)
```

The second object is written to the stream with an invalid digest, produced by calculating the digest for some other data instead of the pickle.

```
# Write an invalid object to the stream
o = SimpleObject('digest does not match')
pickled_data = pickle.dumps(o)
digest = make_digest('not the pickled data at all')
header = '%s %s' % (digest, len(pickled_data))
print '\nWRITING:', header
out_s.write(header + '\n')
out_s.write(pickled_data)

out_s.flush()
```

Now that the data is in the StringIO buffer, we can read it back out again. The first step is to read the line of data with the digest and data length. Then the remaining data is read (using the length value). We could use pickle.load() to read directly from the stream, but that assumes a trusted data stream and we do not yet trust the data enough to unpickle it. Reading the pickle as a string collect the data from the stream, without actually unpickling the object.

```
# Simulate a readable socket or pipe with StringIO
in_s = StringIO(out_s.getvalue())

# Read the data
while True:
    first_line = in_s.readline()
    if not first_line:
        break
    incoming_digest, incoming_length = first_line.split(' ')
    incoming_length = int(incoming_length)
    print '\nREAD:', incoming_digest, incoming_length
    incoming_pickled_data = in_s.read(incoming_length)
```

Once we have the pickled data, we can recalculate the digest value and compare it against what we read. If the digests match, we know it is safe to trust the data and unpickle it.

```
actual_digest = make_digest(incoming_pickled_data)
print 'ACTUAL:', actual_digest

if incoming_digest != actual_digest:
    print 'WARNING: Data corruption'
else:
    obj = pickle.loads(incoming_pickled_data)
    print 'OK:', obj
```

The output shows that the first object is verified and the second is deemed "corrupted", as expected:

```
$ python hmac_pickle.py
```

WRITING: 387632cfa3d18cd19bdfe72b61ac395dfcdc87c9 124

WRITING: b01b209e28d7e053408ebe23b90fe5c33bc6a0ec 131

READ: 387632cfa3d18cd19bdfe72b61ac395dfcdc87c9 124

ACTUAL: 387632cfa3d18cd19bdfe72b61ac395dfcdc87c9
OK: digest matches

READ: b01b209e28d7e053408ebe23b90fe5c33bc6a0ec 131
ACTUAL: dec53ca1ad3f4b657dd81d514f17f735628b6828
WARNING: Data corruption

Hash-based message authentication code (HMAC) is a mechanism for calculating a message authentication code involving a hash function in combination with a secret key. This can be used to verify the integrity and authenticity of a message.

Unlike the previous authentication methods there isn't, as far as we can tell a standard way to do this within HTTP, that said as this is the main authentication method used by Amazon Web Services it is very well understood, and there are a number of libraries which implement it. To use this form of authentication you utilize a key identifier and a secret key, with both of these typically generated in an admin interface (more details below).

It is very important to note that one of the BIG difference with this type of authentication is it signs the entire request, if the content-md5 is included, this basically guarantees the authenticity of the action. If a party in the middle fiddles with the API call either for malicious reasons, or bug in a intermediary proxy that drops some important headers, the signature will not match.

The use HMAC authentication a digest is computed using a composite of the URI, request timestamp and some other headers (depending on the implementation) using the supplied secret key. The key identifier along with the digest, which is encoded using Base64 is combined and added to the authorisation header.

The following example is from Amazon S3 documentation.

```
"Authorization: AWS " + AWSAccessKeyId + ":"  + base64(hmac-sha1(VERB + "\n"
        + CONTENT-MD5 + "\n"
        + CONTENT-TYPE + "\n"
        + DATE + "\n"
        + CanonicalizedAmzHeaders + "\n"
        + CanonicalizedResource))
```

Which results in a HTTP request, with headers which looks like this.

```
PUT /quotes/nelson HTTP/1.0
Authorization: AWS 44CF9590006BF252F707:jZNOcbfWmD/A/f3hSvVzXZjM2HU=
Content-Md5: c8fdb181845a4ca6b8fec737b3581d76
Content-Type: text/html
Date: Thu, 17 Nov 2005 18:49:58 GMT
X-Amz-Meta-Author: foo@bar.com
X-Amz-Magic: abracadabra
```

Note the AWS after the colon is sometimes known as the service label, most services I have seen follow the convention of changing this to an abbreviation of their name or just HMAC.

If we examine the Amazon implementation closely a few advantages become obvious, over normal user names and passwords:

1. As mentioned HMAC authentication guarantees the authenticity of the request by signing the headers, this is especially the case if content-md5 is signed and checked by the server AND the client.
2. An admin can generate any number of key pairs and utilise them independent of their Amazon credentials.
3. As noted before these are computed values and can be optimised to be as large as necessary, Amazon is using 40 character secrets for SHA-1, depending on the hash algorithm used.
4. This form of authentication can be used without the need for SSL as the secret is never actually transmitted, just the MAC.
5. As the key pairs are independent of admin credentials they can be deleted or disabled when systems are compromised therefor disabling their use.

As far as disadvantages, there are indeed some:

1. Not a lot of consistency in the implementations outside of the ones which interface with Amazon.
2. Server side implementations are few in number, and also very inconsistent.
3. If you do decide to build your own be advised Cryptographic APIs like OpenSSL can be hard to those who haven't used them directly before, a single character difference will result in a completely different value.
4. In cases where all headers within a request are signed you need to be VERY careful at the server or client side to avoid headers being injected or modified by your libraries (more details below).

1. When writing the API ensure you check your request on the wire to ensure nothing has been changed or "tweaked" by the HTTP library you're using, mine added a character encoding attribute to the Content-Type.
2. Test that order of your headers is correct on dispatch of the request as well, libraries my use an hash map (natural ordered), this may break your signature depending on the implementation. In the case of Amazon they require you to sort your "extra" headers alphabetically and lower case the header names before computing the signature.
3. Be careful of crazy Ruby libraries that snake case your header names (yes this is bad form) before presenting them to your code as the list of header names.
4. When debugging print the canonical string used to generate the signature, preferably using something like ruby inspect which shows ALL characters. This will help both debugging while developing, and to compare against what the server side actually relieves.
5. Observe how various client or server APIs introduce or indeed remove headers.

From a security stand point a couple of basic recommendations.

1. Use content MD5 at both ends of the conversation.
2. Sign all headers which could influence the result of the operation as a minimum.
3. Record the headers of every API call that may have side affects, on most web servers this can be enabled and added to the web logs (again ideally this would be encoded like what ruby inspect does).

Skipjack Transactions

Encrypcurrency is the digital encryption dime as virtual currency in the form of superencryp block cipher with Key Exchange Algorithm and a chain of digital signatures. Each owner transfers the dime to the next by digitally signing a hash MAC of the previous transaction and the public key with Key Exchange Algorithm of the next owner and adding these to the end of the dime transactions. A payee can verify the signatures based on FCN registry and HMAC verification to verify the chain of ownership. Skipjack transactions based on the 3 basic protocol as the rule message of internet protocol. All data in internet, in telephone, mobile phones, satellite to transmit there will have to follow 3 basic rule protocol in order our data or our voice in telephone to be secured. In the world of communication technology especially in digital we need to have 3 basic protocols:

1. Data hashing / authentication

2. Key exchange algorithm / assymetric

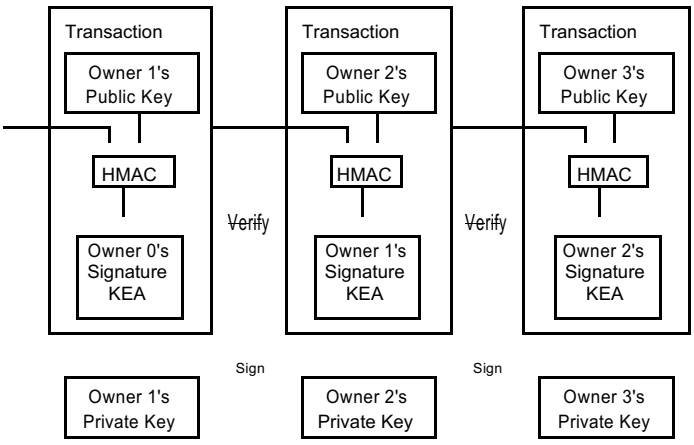
3. Data security encryption

The different of Skipjack compare to other virtual currency in the market is that Skipjack is covered with full suite protocols. Currently others only based on1 basic protocol which is data hashing and digital signatures. Most protocols like data hashing SHA is not an encryption. It is only to digest and used for comparison. Meaning it is only converting data password without encryption based on hash function and digital signatures.

In modern internet, all communications must have 3 protocols same as your internet today. Skipjack algorithm is build based on the 3 basic protocols. The protocols work the same as internet TCP/IP protocols. Skipjack may have the same as becoming internet 2.0. Skipjack protocols can take 400 billion years to break if from one single linear computer.

Skipjack Key Exchange Algorithm is much secured by nature compare to DES or AES in your modern internet protocol TCP/IP.Skipjack may the new rise of the new technology in future.

Architecture Skipjack Encrypcurrency Protocol:



A and B need not complete the key-exchange protocol before exchanging transactions. In this protocol, A sends B the transaction, *M*, without any previous key exchange protocol: FCN signs both A's and B's public keys. The signed keys include a signed certification of ownership. When A and B receive the keys, they each verify FCN's signature. Now they know that the public key belongs to that other person. The key exchange protocol can then proceed. A generates a random session key, *K*, and encrypts *M* using *K*.

$$E_{\mathrm{K}}(M)$$

A gets B's public key from the database.

A encrypts *K* with B's public key.

$$E_{\mathrm{B}}(K)$$

A sends both the encrypted transaction and encrypted session key to B.

$$E_{\mathrm{K}}(M),\ E_{\mathrm{B}}(K)$$

For added security against man-in-the-middle attacks, A can sign the transaction. B decrypts A's session key, *K*, using his private key.B decrypts A's transaction using the session key.

This hybrid system is how public-key encrypcurrency is used in a communications and transactions system. It can be combined with digital signatures, timestamps, and any other security protocols.

The receiver can't verify that one of the owners did not double- spend the coin. The successful verification of a MacTag does not completely guarantee that the accompanying text is authentic; there is a slight chance that an adversary with no knowledge of the HMAC key, K, can present a (MacTag, text) pair that will pass the verification procedure. The solution is using a trusted central authority Fiestel Core Network (FCN) that checks every transaction for double spending. Each transaction, the transactions verified by the "checker" and FCN by applying double hash address to verify are trusted not to be double-spent. The fate of the entire money system depends on the FCN and the checker with every transaction having to go through them, just like a bank.

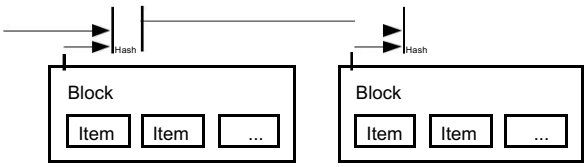
We need a way for the payee to know that the previous owners did not sign any earlier transactions. For our purposes, the earliest transaction is the one that counts, so we don't care about later attempts to double-spend. The only way to confirm the absence of a transaction is to be aware of all transactions. In the "Checker" based model, the public checker was aware of all transactions and decided which arrived first. To accomplish this without a trusted party, transactions must be publicly announced, and we need a system for participants to agree on a single history of the order in which they were received. The payee needs proof that at the time of each transaction, the majority of nodes agreed it was the first received. This is done by applying double HMAC from the "Checker"

Timestamp Server

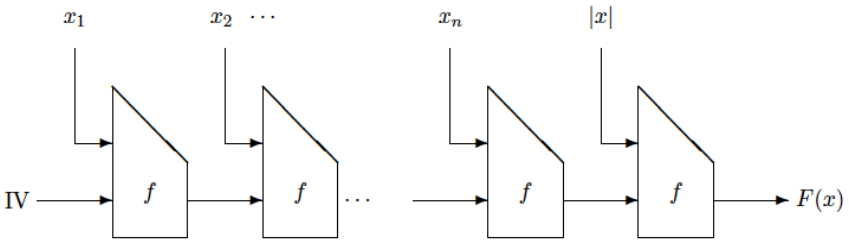
A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash.The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.In encrypcurrency we have improved by arbitrated solution.

Improved Arbitrated Solution
One-way hash functions and digital signatures can clear up most of these problems easily:
(1) A produces a one-way hash of the document.
(2) A transmits the hash to FCN.
(3) FCN appends the date and time he received the hash onto the hash and then digitally signs the result.
(4) FCN sends the signed hash with timestamp back to A.

This solves every problem but the last. A no longer has to worry about revealing the contents of document; the hash is sufficient. FCN no longer has to store copies of the document (or even of the hash), so the massive storage requirements and security problems are solved (remember, one-way hash functions don't have a key). A can immediately examine the signed timestamped hash received in step (4), so will immediately catch any transmission errors. The only problem remaining is that A and FCN can still collude to produce any timestamp they want.



Double HMAC Verification



Encrypcurrency is using HMAC authentication. HMAC is an algorithm for producing a keyed message authentication code using a hash function. (The output of the HMAC algorithm is also referred to as an HMAC.) HMAC takes as input a message and key, and produces a fixed-length digest output which can be sent with the message. Anyone who knows the key can repeat the algorithm and compare their calculated HMAC with one they have received, to verify a message originated from someone with knowledge of the key and has not been tampered with. The idea is simple enough - the problem comes when performing the comparison.

The standard algorithm to compare byte arrays for equality looks like:

```
if(byteArrayA.length != byteArrayB.length) { return false; }
for(int i = 0; i < byteArrayA.length; i++) {
  if(byteArrayA[i] != byteArrayB[i]) { return false; }
}
return true;
```

The problem with this algorithm is that the more bytes match in the two arrays, the more comparisons are performed, and the longer it takes to return. The consequence is that an attacker who can measure this difference in timing can submit guesses at a valid HMAC for an arbitrary text, and check the correctness of the guesses one byte at a time. The attacker submits guesses for the first byte until the value is discovered which takes longer to return an error, then that value is "locked in" and guessing can proceed to the next byte. This makes brute force trivial, even for arbitrarily long HMAC values. The timing differences are very subtle and difficult to measure in some contexts, but successful attacks have been mounted against real systems using this technique.

We use a constant-time comparison function to avoid this flaw when verifying HMACs. This is a good recommendation for native code, but building a true constant-time function is not as simple as it sounds for the .Net CLR, JVM or other high level languages. We wrote test code in Java and C#, and found that often source code which reads as constant time doesn't run that way. The compiler, the JIT, and runtime optimizers on these platforms do not generally recognize timing as a program semantic.

In light of this, we've begun recommending a different solution: **double HMAC verification.** After receiving and calculating an HMAC for a given message, **simply apply an additional round of HMAC** to both byte arrays before comparing them. Instead of removing the timing side channel, this deterministically randomizes the bits of an attacker's guess in a way he or she can't predict. Without knowledge of the actual bytes being compared, the guesses can't be adapted and the timing side channel is useless.

We like this solution for a few reasons:

- It is easy to implement. Usually it requires only two lines of code be added to an existing verification routine.
- It is still relatively fast. HMAC is a speedy operation, and the additional round starts with an input already at the block size of the hash algorithm.
- Its security derives from the fundamental cryptographic properties of HMAC, so is not dependent on runtime behavior, optimizations, or the lack thereof.

Here's some C# code to implement double HMAC verification:

```
public void validateHMACSHA256(byte[] receivedHMAC, byte[] message, byte[] key)
  {
HashAlgorithm hashAlgorithm = new HMACSHA256(key);

// size and algorithm choice are not secret; no weakness in failing fast here.
if(receivedHMAC.Length != hashAlgorithm.HashSize / 8){
   Throw new CryptographicException("HMAC verification failure.");
}
      byte[] calculatedHMAC = hashAlgorithm.ComputeHash(message);
      // Now we HMAC both values again before comparing to randomize byte order.
      // These two lines are all that is required to prevent many existing implementations
      // vulnerable to adaptive chosen ciphertext attacks using the timing side channel.
      receivedHMAC   = hashAlgorithm.ComputeHash(receivedHMAC);
      calculatedHMAC = hashAlgorithm.ComputeHash(calculatedHMAC);

      for (int i = 0; i < calculatedHMAC.Length; i++)  {
        if (receivedHMAC[i] != calculatedHMAC[i])  {
           throw new CryptographicException("HMAC verification failure.");
        }
      }
 }
```

So why re-use HMAC instead of using some other pseudo-random function? Because it does not to make the resulting construct any weaker. As a reduction ad absurdum, most people would agree that CRC-16 does not have the necessary properties as a PRF to make adaptive guessing infeasible. Is MD5 a "good enough" PRF to foil timing attacks? Probably. Is SHA-1? Almost certainly. But what if you have regulatory requirements to use only SHA-512 in your product? Is SHA-1 still "good enough"? Reusing the HMAC function removes the cost of this choice and keeps the solution cryptographically agile. [http://msdn.microsoft.com/en-us/magazine/ee321570.aspx]

And why use a keyed hash, instead of just the raw hash? That one's a little trickier. A robust PRF would frustrate the attack, but the HMAC construction itself has some properties in this regard that are independent of the hash algorithm it is implemented with. For example, Mihir Bellare showed in 2006 that HMAC is a secure MAC under the sole assumption that the underlying compression function is a privacy-preserving MAC, so it is possible to have a secure HMAC using a weaker-than-PRF compression function. [http://academic.research.microsoft.com/Paper/2293939.aspx] We choose to recommend re-applying the full, keyed HMAC, in case there exist or are discovered attacks which violate the PRF assumption of a compression function without invalidating the security of an HMAC constructed with that function. At the end of the day, the cost saved in the analysis again outweighs the runtime cost for most applications.

So, why use double HMAC? Don't use it to save a few CPU cycles. Use it because it is the simplest, safest and most broadly correct security engineering decision. The CPUs will mostly take care of themselves.

Classification

FCN Abstract

An encrypsystem describes the system where two or more individuals communicate in a secret manner over a public channel. The contents of information shared on a public network need to be protected against unauthorized access using a more secured technique of encryption and decryption. A cipher is the heart of a encrypsystem, which specifies the policy for encryption and decryption. The encryption policy takes the plaintext and a predetermined key, and produces a ciphertext which hides the true meaning of the original plaintext. The decryption policy takes another key, which may be different from the encryption key, and recovers the plaintext from the ciphertext. In this FCN design, an encrypsystem is developed using Skipjack algorithm for key exchange purpose during information sharing on a public network. The proposed system provides a secure platform for encrypting and decrypting user's key. This key could be used by a symmetric algorithm for file encryption and decryption. The system is found capable of securing user's key from illegal access by an unauthorized person.

FCN Introduction

Computer has been primarily used by university researchers in some years back majorly for information transfer among academics and organizations for resource sharing purpose. These resources include printers and scanner. At this time, security was not on the priority, and such shared information and resources do not pose any security risk to sender or receiver. Nowadays, the use of computer has gone beyond their early intentions. Today’s information technology security environment consists of highly interactive and very powerful computing devices and interconnected systems of systems across global networks to share resources among individuals.

This has led to the need for tight security to data and information as they are transported across public networks. The security of communications and commerce in a digital age relies on the modern incarnation of the ancient art of codes and ciphers. Underlying the birth of modern cryptography is

a great deal of fascinating mathematics, some of which has been developed for cryptographic applications, but much of which is taken from the classical mathematical rules . Cryptography is the science concerned with linguistic and mathematical techniques for securing information.

Cryptography has been concerned solely with encryption which involved converting information from its normal, comprehensible form into an incomprehensible format, thus rendering it useless without secret knowledge Cryptography is used in application represent in technological advanced societies; examples include the security of email, ATM cards, computer passwords and electronic commerce which all depend on cryptography. Indeed, the principal goal in the design of any cryptographic system is security.

An encrypsystem describes a way for two or more parties to communicate in secret over a public channel. The content of the communication, which may be human language or anything else, is called the plaintext. The heart of a encrypsystem is a cipher, which specifies rules for encryption and decryption. The encryption rule takes the plaintext and a pre-determined key, and produces a ciphertext which hides the content of the original plaintext. The decryption rule takes another key, possibly different from the encryption key, and recovers the plaintext from the ciphertext. There are two basic types of cryptosystems: symmetric key (private key) systems and public key (asymmetric) systems. Public key systems are much slower than symmetric systems, but symmetric systems require key agreement through an existing secure channel, hence, the two systems can be combined.

FCN and Superencryp Block Design

Encryption keys need to be pre-distributed among the devices that want to communicate with each other. Algorithms for deploying these pre-shared keys need to be implemented, which must support requirements based on the setup of its FCN. Key distribution schemes should mainly provide authenticity to provide a way to identify nodes, scalability, and flexibility to allow adding of new nodes at every time. Authenticity, scalability, and flexibility lead to several basic attack scenarios key distribution schemes have to protect against. An attacker could, for example, compromise certain parts of the network and replicate those nodes to take over the entire network. Key distribution schemes should resist against those attacks by preventing replication. When bad behavior of single nodes is detected, those nodes should be excluded, by revoking them using the key management scheme. Furthermore, it should be possible to deploy new nodes into existing key management infrastructures and to retain secrecy of benign nodes, when single nodes are taken over (Resilience).
Several key pre-distribution schemes were proposed to fulfill these requirements, which are based on assumptions regarding the skipjack network. We will present fundamental problems, the most common schemes, and show possible weaknesses.

Fundamentals

The simplest key pre-distribution form is the pre-deployment of a single network wide key on all nodes in the network. Besides small storage requirements, this scheme apparently does not prevent any of the three discussed attacks as a single compromised node results in a compromised network.
In a pairwise key distribution scheme, a node is deployed with n - 1 keys to communicate with every other node. This pairwise encryption and authentication as every key belongs uniquely to two nodes. Pairwise pre-distribution also complies with Resilience, because a take over of nodes does not reveal secret information about other nodes due to secret keys. This scheme only makes sense in small networks as each new node requires a new key to be stored by all other nodes, resulting in a high memory usage. A superior sink node with a larger memory and higher transmission rates can be used as Key Distribution Center (KDC). A KDC can be queried for session keys used one time for a speciftc communication between nodes.

Random Key Pre-distribution

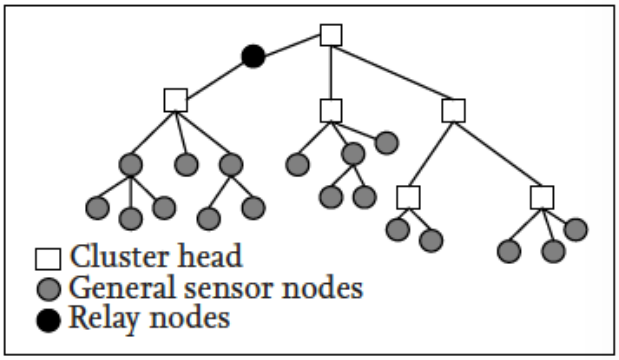
Eschenauer and Gligor presented a random key pre-distribution protocol to over- come the problem of storing $N - 1$ keys on every participant. The distribution scheme works as follows:

1. A large pool of $|S|$ key-identifier pairs is generated.
2. K key-identifier pairs are randomly taken out of this pool, where $K << N$ and N is the number of nodes in the FCN and saved as a keyring. Every node gets its own keyring assigned.
3. Based on the identifiers, two nodes can agree upon one shared key, while the probability is high that they both share at least one matching key. If no shared key is available, the nodes perform path-key discovery. This means a third party node will be searched that shares keys of both communicating nodes.

The scheme successfully optimizes regarding storage space and is also scalable, as the size of the underlying key pool and the number of chosen keys can be adaptively chosen according to the size of the network. The downsides of this schemes are that it provides no form of multicasting messages because between the sender and every recipient a key has to be negotiated. There is also no method that would provide strong revocation of compromised nodes and freshness of keys.

Schemes Based on Random Key Pre-distribution

Several more complex protocols were proposed that are based on the random pre-distribution model and published a model extending the concept using Blom's key matrix instead of individual keys. A matrix $M_{n\,x\,n}$ is generated, where nodes can select keys from by looking at specifc cells. If node i and j want to communicate, i looks at M_{ij} and j looks at M_{ji} to select the same shared key. In this scheme n matrices are generated and on each node only randomly selected matrices are stored, like in the basic random key pre-distribution scheme. Their scheme more robustness and security, because more nodes need to be compromised compared to the basic random pre-distribution scheme. This advantages comes to the cost of more computational overhead and memory consumption.
Even more complex protocols like LEAP and SHELL were published .They introduce key types, like individual, group, cluster, and pairwise keys, where each key type is distributed and used. They fulfill most requirements are flexible, but require complex implementations. Consequently much memory space is needed for their implementation and keys. Besides their memory usage, their implementation will likely contain bugs due to their high complexity.



Hierarchical structure proposed

Hierarchical FCN Group Key Management Schemes

A hierarchical FCN management scheme consists of groups sharing the same group key, ordered hierarchically based on their computing power or other factors like access rights. The scheme consists of cluster heads, having many general nodes as children, arranged in a tree structure .A partial key for each node is computed as follows :

1. All leaf nodes generate their keys randomly.
2. The parents of these leaf nodes compute their keys based on the child nodes and the given function $f(k_1,k_2)=\alpha^{k1\oplus k2} \mod p$, where k_1,k_2 are keys and p is prime with $k < p$ for both k.
3. When the corresponding tree is binary, a partial key is calculated by the following equation, where the index of K defines the position in the tree by depth and row.

$$K_{l,v}=f(K_{l+1,2v}, K_{l+1,2v+1})$$

4. In situations where the tree is not binary a slightly modiftcation is added, where $m = n_{siblings}\ 2$ and $n_{siblings}$ deftnes the number of siblings in this row.

$$K_{l,v}=f(K_{l+1,2v+m}, K_{l+1,2v+1+m})$$

Group key computation is done by utilizing multi-party Diffie-Hellman and TGDH. A cluster head node can initiate a group key computation, where every all partial keys are gathered bottom-up and collected by the cluster head.

1. All leaf nodes generate a partial key g^S, where $(g) = Z_p^*$, p prime, and S is a randomly generated number. These partial keys are send to their parent nodes.

2. Nodes, which are not cluster heads and not leafs, receive partial keys from all their n children and sums them up together with their own partial key S to generate an intermediate key IK.

$$IK = g^{S1} + \cdots + g^{Sn} + g^S = g^{S1 \cdot \cdots \cdot Sn \cdot S}$$

3. Finally, the cluster head receives all intermediate keys and sums them up again together with its own partial key S, resulting in the group key K.

4. This group key is encrypted with a one-time symmetric key, only available in deployment phase, and broadcasted to all nodes of this group. Afterwards the symmetric key is discarded and the group key is used.

A new group key is calculated, when a cluster head broadcasts an encrypted authenticated message containing the instruction to refresh the group key. This could also contain the instruction to remove a compromised node, which implements revocation.
In addition to these algorithms between intra-cluster group keys and inter-cluster group keys. When the cluster head broadcasts the key the first time, blind factors are added, which are unique numbers assigned to individual nodes. When distributing the group key, each node can recognize its own blind factor and replace it with their own real factor. This prevents attackers from eavesdropping the group key in deployment phase.
This scheme is flexible, because it is possible to refresh group keys and revoke nodes. However, revocation only works to a certain degree, because cluster heads can only revoke leafs or complete branches. The downside is that it is assumes that a cluster head does not get compromised, which makes it a worthy target. It is also quite elegant and simpler to implement than over-engineered schemes like SHELL improved upon such schemes to reduce the rekeying overhead.

ECC Based Key Management

Key pre-distribution schemes do not require large amount of processing power or RAM, but are much less flexible than schemes based on public key cryptography. In the past, most networks were deployed with pre-distributed keys, lacking important requirements like fast revocation of keys and secure integration of newly deployed nodes. RSA based public key algorithms were omitted due to their seemingly large power consumption while performing underlying multiplication methods. New implementations of Elliptic Curve Cryptography (ECC) were done to provide usable public key cryptography with low requirements. ECC is based upon the Elliptic Curve Discrete Logarithm Problem (ECDLP), which states that it is hard to find a discrete logarithm of a elliptic curve element with a publicly known base point. According to the basic definitions are as follows.

Discrete logarithm problem The discrete logarithm problem is to compute l such that $\beta = \alpha^l$. α is a generator for a multiplicative group $(\alpha) = G$ and $\beta \in G$.

Elliptic Curve Discrete Logarithm Problem Compute $l \in [0, n-1]$ such that $Q = lP$. Let P be a point of order n and $Q \in (P)$

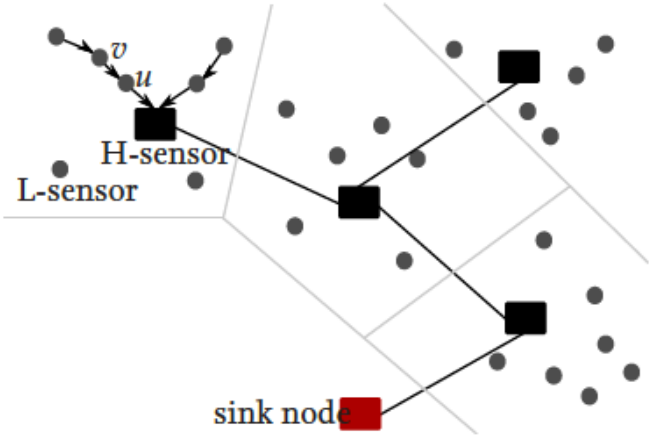
Gura et al. have done implementations of public key cryptography for RSA and ECC for sensor nodes based on ATmega128 and CC1O1O microcontroller. Their implementations are highly optimized regarding memory access, as this is the most limiting factor on these small processors. The key size for ECC keys can be much lower than RSA based keys, e.g., a 1024 bit RSA key pair provides as much security as an 160 bit ECC key pair. This is due to the fact that RSA trapdoor function is based on the hardness of integer factorization and ECC is based on its ECDLP. While sub-exponential algorithms exits to solve integer factorization, known algorithms to solve ECDLP scale exponentially. Their ECC point-multiplication using 160 bit, implementing SECGs standardized elliptic curves in conjunction with their optimized multiplication, is by a factor of two faster than RSA-1024. As an example by absolute numbers, RSA-1024 private key operation needed 10.99 s while a 512 bit montgomery exponentiation for ECC needed 5.37 s. ECC-240 outperforms RSA-2048 by a much more higher degree as shown in their paper.

Their evaluation has shown that it is indeed possible to use public key algorithms without hardware acceleration on microcontrollers intended for nodes. As recommended key sizes by NIST for 2011-2030 are RSA-2048 and ECC-240, ECC should be favored due to their small key sizes and low runtime of its algorithms. Besides implementing a hybrid multiplication algorithm, they chose Koblitz curves as these are less expensive due to their appearance without point doubling. Their implementation is done for TinyOS and written in C/nesC with some optimizations using assembly, generated by utility programs for better portability. It is based on MIRACL[t], a library that provides all necessary primitives for ECC and was optimized for embedded platforms. The ECC implementation is based on NIST recommended k1G3 Koblitz curves over $GF(2^{162}$ and $y^2 = x^3\ 3x + 157$ with $p = 2^{160}\ 2^{112} + 2^{64} + 1$ over $GF(p)$. To speed up the point multiplication in elliptic curves, which are defined as sP with s as an integer and P a point of the curve, additional memory is utilized. Their implementation was evaluated on two sensor nodes, the MICA2 (ATmega128L) and Tmote Sky (MSP-430). Computations for the binary field were between 1.04 s and 2.16 s and 0.72s to 1.27 s in the prime field. Thus, NanoECC demonstrates an efficient implementation usable in practice.

Routing-Driven Key Management

Sensor nodes often only communicate with their communication-neighbors to route data to the sink node. A node v is a communication-neighbor (c-neighbor) of a node u if v is on a route from u to the sink node. Because of this many-to-one communication pattern, nodes do not need to obtain shared keys for all other nodes in the network. Thus, while in key establishment phase, shared keys only need to be generated for every node and its neighbors.

[t]MIRACL can be found on http://www.shamus.ie/index.php?page=elliptic-curves



Routing-driven key management: Clusters with many low-end sensors (L-sensors), controlled by single high-end sensors (H-sensors).

Between high-end (H-sensors) and low-end sensors (L-sensors) with hardware configurations. A small number of H-sensors exist in a network, which are equipped with tamper-resistant hardware, preventing an adversary to extract keys, data, or stored code. Where as L-sensors are low cost devices without such protection. Both sensor types are location aware. As depicted in Figure3, H-sensors form a basis station for clustering many L-sensors together, and are used as gateways to route traffic to the sink node of the whole FCN. We will introduce the centralized key establishment described in the paper and omit the distributed scheme.

Centralized Key Establishment

1. A server generates public/private key pairs (K_u^R,K_u^U). Every L-sensor is preloaded with its private key, whereas every H-sensor also contains the public keys of all L-sensors.
2. Each H-sensor stores a common ECC public/private key pair (K_H^U, K_H^R) for authenticating broadcasts using Elliptic Curve Digital Signature Algorithm (ECDSA). The public part is also stored on every L-sensor to verify these broadcasts. Additionally, H-sensors are deployed with a symmetric key K_H used for communicating among H-sensors.
3. Inside a cluster, each L-sensor sends key-request messages to their H-sensor, which includes their unique ID and location. The H-sensor generates shared- keys $K_{u,v}$ for all L-sensors in its cluster for sensor u and its c-neighbor v, using its public key K_u^R. This shared-keys are encrypted and send by unicast to u.
4. u decrypts the message and stores the shared key for communication between u and v.

Remarks

Revocation is done by H-sensors, which should also detect malicious nodes. A revocation message is signed using ECDSA and contains the ID of the compromised node. This message is sent to all L-sensors of this cluster, which can check whether they communicate with a malicious node or not.
The scheme implements revocation and resilience in a feasible way. Their performance evaluation also looks promising considering the amount of L-sensors involved in the setup. This key management requires a well thought out network structure, location aware nodes, and H-nodes with tamper-resistant hardware, which are quite high requirements.

Conclusion

In this paper, we presented selected security primitives for FCN networks. Software implementations of encryption modules and fault attacks against hardware implementations were discussed. Key pre-distribution schemes were depicted with their advantages and disadvantages. As a promising public key technique, ECC could solve some problems pre-distribution schemes could not solve perfectly and is thus viewed as FCNs future key distribution implementation. Several hard issues remain unsolved. The problem of adding nodes to an existing network is difficult even with public key cryptography, as every node in this distributed network as to be informed about newly added and accepted public keys. This can partially be solved by defining the sink node as a trusted third party, as done in Identity Based Cryptography. Revocation is even harder, because of several pitfalls. Firstly, a node needs to be identified malicious node, possibly overtaken by an adversary. Only a few papers deal with this issue, as it is impossible to detect malicious nodes, when they behave correctly and just capture traffic. Secondly, the revocation packet needs to be routed to every node quickly, although malicious nodes could work as black holes, preventing the distribution. These problems exist due to the distributed nature of wireless sensor networks. Furthermore, physical attacks against hardware implementations, as discussed need to be taken more seriously. Many FCN are deployed in areas without any additional surveillance, which makes fault attacks possible and leave them undetected.

SKIPJACK and Key Exchange Algorithm (KEA)

Introduction

This whitepaper covers implementation of the Skipjack algorithm and Key Exchange Algorithms in C using some assembly code routines, if necessary, to optimize the encryption and decryption speeds. Then this implementation is compared to the speed of encryption and decryption for optimized DES implementations available. The Key Exchange Algorithm (KEA) is implemented to be compatible with those used by the FORTEZZA card.

The development was conducted on a Pentium II using GNU C under Red Hat Linux 5.2. The optimized implementation is written in C and is portable across many platforms. The standard C libraries are sufficient for implementing the Skipjack encryption and decryption algorithms. However, in order to implement the KEA they require a library to generate large prime numbers as well as large random numbers. For operations on these large numbers I chose to use the SSLeay libraries. They contain header files and a binary library that can be linked with the final object file to provide functions to perform the above operations.

Overview

This section covers an overview of the SKIPJACK algorithm as well as the Key Exchange Algorithm. Each of the stages of the SKIPJACK encryption and decryption algorithms will be given in detail. Followed by a detailed description of the store and forward KEA for **exchanging data transactions or email with another user**. Implementations on the FORTEZZA card are also covered.

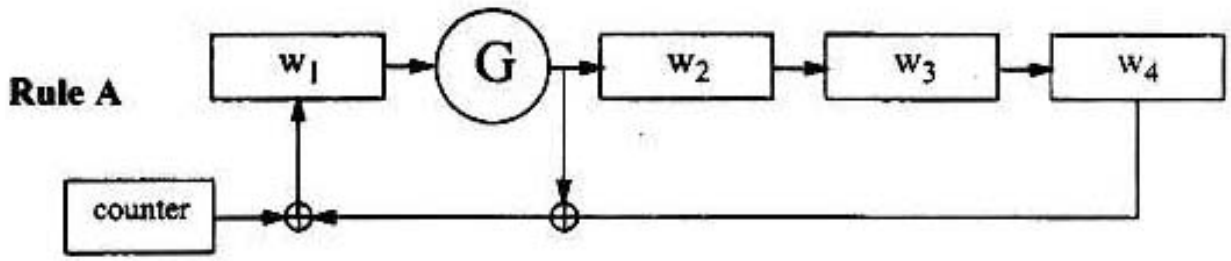
SKIPJACK Algorithm

SKIPJACK operates on a single 64-bit input, as four 16-bit words, producing a 64-bit output using an 80-bit cryptovariable (key). The algorithm works by alternating between two stepping rules (A and B) described below. Each rule is executed eight times before alternating, and there are two alternations for a total of 32 rounds. Each of these stepping rules uses a counter that is initialized with a value of 1 for encryption and 32 for decryption. Both of the stepping rules are reversible functions, allowing for decryption.

Stepping Rule A

The basic structure of the stepping rule A for encryption is given below (the G permutation function is covered later).

a. G permutes w_1,
b. the new w_1 is the exclusive or of the G output, the counter, and w_4,
c. words w_2 and w_3 shift one register to the right; i.e. become w_3 and w_4 respectively,
d. the new w_2 is the G output
e. the counter is incremented by one



The equations for this rule are shown below. The superscript is the counter value:

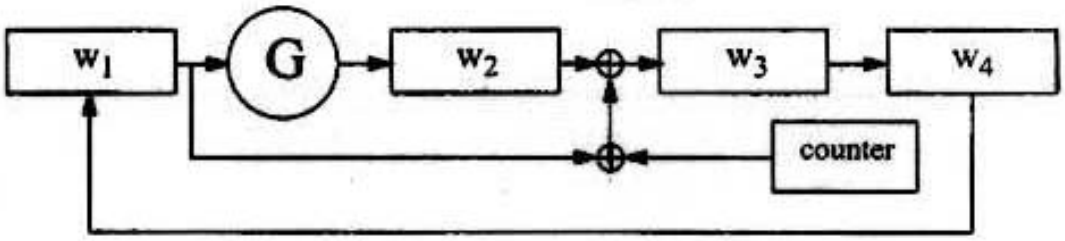
Encryption	**Decryption**
$w_1^{k+1} = G^k\left(w_1^k\right) \oplus w_4^k \oplus counter^k$	$w_1^{k-1} = \left[G^{k-1}\right]^{-1}\left(w_2^k\right)$
$w_2^{k+1} = G^k\left(w_1^k\right)$	$w_2^{k-1} = w_3^k$
$w_3^{k+1} = w_2^k$	$w_3^{k-1} = w_4^k$
$w_4^{k+1} = w_3^k$	$w^{k-1} = w_1^k \oplus w_2^k \oplus counter^{k-1}{}_4$
$counter^{k+1} = counter^k + 1$	$counter^{k-1} = counter^k - 1$

Stepping Rule B

The basic structure of the stepping rule B for encryption is given below (the G permutation function is covered later).

a. G permutes w_1,
b. words w_3 and w_4 shift one register to the right; i.e. become w_4 and w_1 respectively,
c. the new w_3 is the exclusive or of w_1, the counter, and w_2,
d. the new w_2 is the G output
e. the counter is incremented by one

Rule B



The equations for this rule are shown below. The superscript is the counter value:

Encryption	**Decryption**
$w_1^{k+1} = w_4^k$	$w_1^{k-1} = \left[G^{k-1}\right]^{-1}\left(w_2^k\right)$
$w_2^{k+1} = G^k\left(w_1^k\right)$	$w^{k-1} = \left[G^{k-1}\right]^{-1}\left(w_2^k\right) \oplus w_3^k \oplus counter^{k-1}$ 2
$w_3^{k+1} = w_1^k \oplus w_2^k \oplus counter^k$	$w_3^{k-1} = w_4^k$
$w_4^{k+1} = w_3^k$	$w_4^{k-1} = w_1^k$
$counter^{k+1} = counter^k + 1$	$counter^{k-1} = counter^k - 1$

Stepping Sequence

To encrypt data, the counter value is initialized to 1, and the input is divided into four 16-bit words labeled
w_1^0, w_2^0, w_3^0 w_4^0 .Next, 8 steps according to Rule A are performed, followed by 8 steps according to Rule B. After this another 8 steps of Rule A are performed and then completed with another 8 steps of Rule B, for a total of 32 steps. The output is then the results w_1^{32}, w_2^{32}, w_3^{32} w_4^{32}

To decrypt data the counter value is initialized to 32, and the input is divided into four 16-bit words labeled
w_1^{32}, w_2^{32}, w_3^{32} w_4^{32} Next, 8 steps according to Rule B^{-1} are performed, followed by 8 steps according to Rule A^{-1}.After this another 8 steps of Rule B^{-1} are performed and then completed with another 8 steps of Rule A^{-1}, for a total of 32 steps. The output is then the results w_1^0, w_2^0, w_3^0 w_4^0

G Permutation

The permutation function **G** is key dependent four round Feistel structure. The round function uses a fixed byte substitution table (which operates on 8-bit bytes) called the F-table. For encryption the 16-bit input is divided into its upper 8 bits and lower 8 bits as g_1 and g_2, respectively. The final output is then the concatenation of two 8-bit values g_5 and g_6. The general formula for **G**, recursively, is given by:

$$g_i = F\left(g_{i-1} \oplus cv_{4k+i-3}\right) \oplus g_{i-2}$$

Where k is the value of the counter (step number), F is the substitution, and cv_{4k+i-3} , is the $(4k+i-3)^{th}$ byte in the cryptovariable (key) schedule. Since the key is 10 bytes long, the value of 4k+i-3 is taken modular 10, with the key bytes labeled 0 though 9.

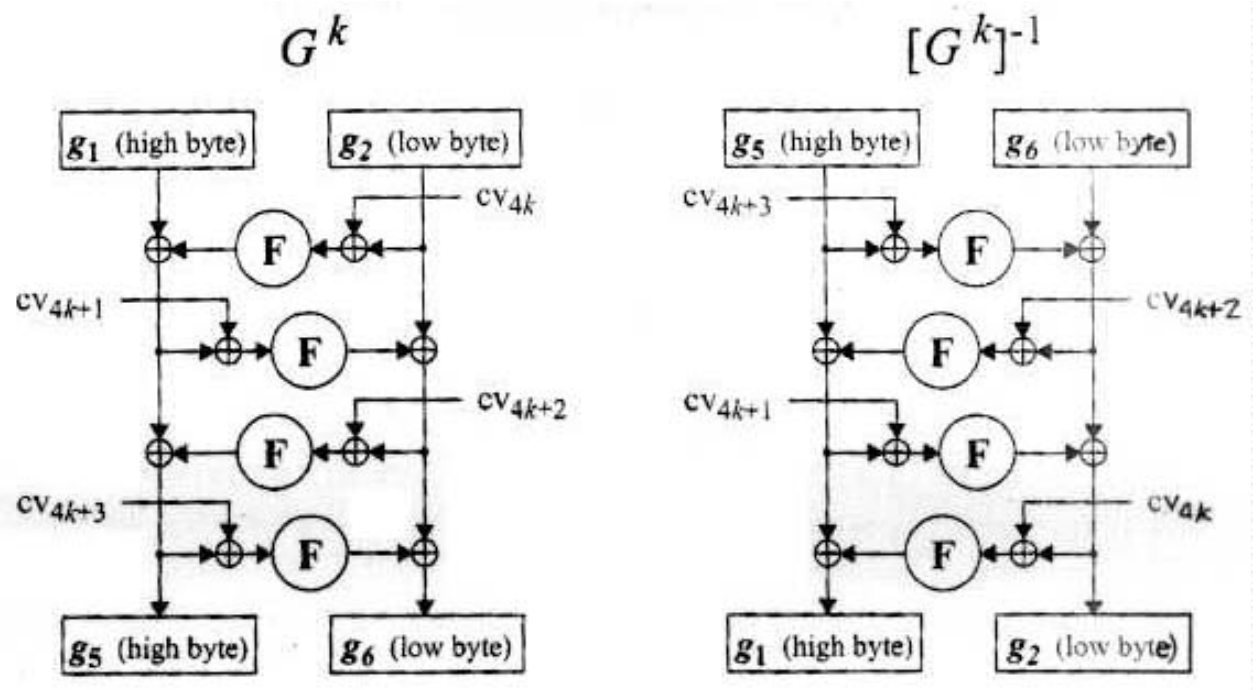
For decryption the 16-bit input is divided into its upper 8 bits and lower 8 bits as g_5 and g_6, respectively. The final output is then the concatenation of two 8-bit values g_1 and g_2. The general formula for $\mathbf{G^{-1}}$, recursively, is given by:

$$g_{i-2} = F\left(g_{i-1} \oplus cv_{4k+i-3}\right) \oplus g_i$$

The expansion of both recursive functions produces:

Encryption	Decryption
$g_3 = F\left(g_2 \oplus cv_{4k}\right) \oplus g_1$	$g_4 = F\left(g_5 \oplus cv_{4k+3}\right) \oplus g_6$
$g_4 = F\left(g_3 \oplus cv_{4k+1}\right) \oplus g_2$	$g_3 = F\left(g_4 \oplus cv_{4k+2}\right) \oplus g_5$
$g_5 = F\left(g_4 \oplus cv_{4k+2}\right) \oplus g_3$	$g_2 = F\left(g_3 \oplus cv_{4k+1}\right) \oplus g_4$
$g_6 = F\left(g_5 \oplus cv_{4k+3}\right) \oplus g_4$	$g_1 = F\left(g_2 \oplus cv_{4k}\right) \oplus g_3$

Schematically **G** and $\mathbf{G^{-1}}$ are shown in the figure below:



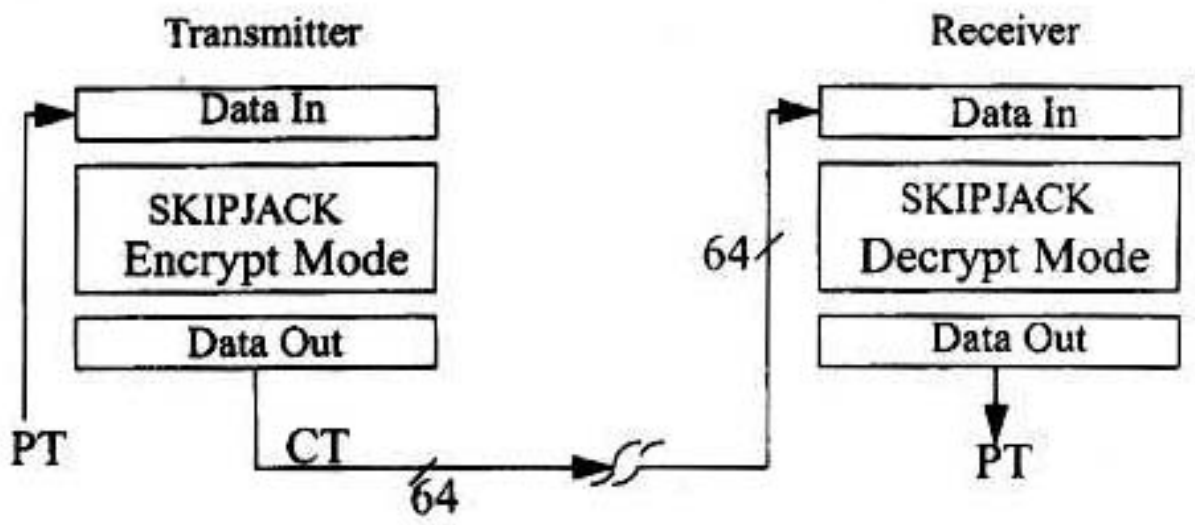
F Table

The SKIPJACK F-table is shown below in hexadecimal notation. The upper 4 bits of the input index the row and the lower 4 bits index the column. For example, F(6D) = 98.

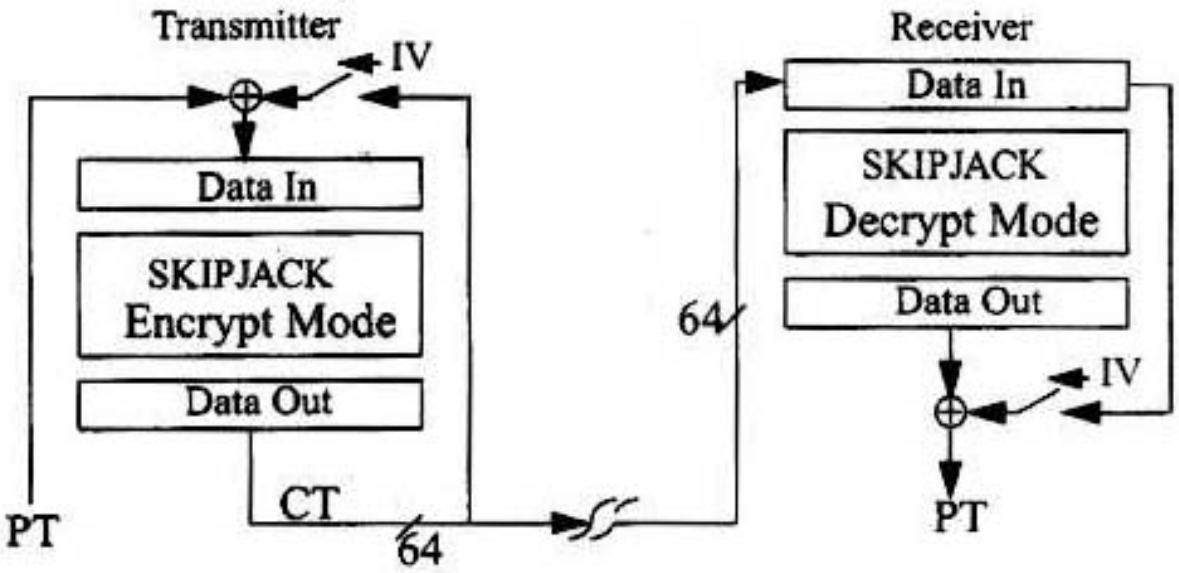
	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	a3	d7	09	83	f8	48	f6	f4	b3	21	15	78	99	b1	af	f9
1x	e7	2d	4d	8a	ce	4c	ca	2e	52	95	d9	1e	4e	38	44	28
2x	0a	Df	02	a0	17	f1	60	68	12	b7	7a	c3	e9	fa	3d	53
3x	96	84	6b	ba	f2	63	9a	19	7c	Ae	e5	f5	f7	16	6a	a2
4x	39	b6	7b	0f	c1	93	81	1b	ee	b4	1a	ea	d0	91	2f	b8
5x	55	b9	da	85	3f	41	bf	e0	5a	58	80	5f	66	0b	d8	90
6x	35	d5	c0	a7	33	06	65	69	45	00	94	56	6d	98	9b	76
7x	97	Fc	b2	c2	b0	fe	db	20	e1	Eb	d6	e4	dd	47	4a	1d
8x	42	Ed	9e	6e	49	3c	cd	43	27	d2	07	d4	de	c7	67	18
9x	89	Cb	30	1f	8d	c6	8f	aa	c8	74	dc	c9	5d	5c	31	a4
Ax	70	88	61	2c	9f	0d	2b	87	50	82	54	64	26	7d	03	40
Bx	34	4b	1c	73	d1	c4	fd	3b	cc	Fb	7f	ab	e6	3e	5b	a5
Cx	ad	04	23	9c	14	51	22	f0	29	79	71	7e	ff	8c	0e	e2
Dx	0c	Ef	bc	72	75	6f	37	a1	ec	d3	8e	62	8b	86	10	e8
Ex	08	77	11	be	92	4f	24	c5	32	36	9d	cf	f3	a6	bb	ac
Fx	5e	6c	a9	13	57	25	b5	e3	bd	a8	3a	01	05	59	2a	46

SKIPJACK Modes of Operation

SKIPJACK can be used in four modes of operation, which is a subset of the FIPS-81 description of modes of operation for DES. These include Output Feedback (OFB) modes, Cipher Feedback (CFB) modes, Codebook, and Cipher-block Chaining (CBC). The FORTEZZA card implements the Codebook and CBC modes of operation for SKIPJACK.



SKIPJACK Codebook



SKIPJACK Cipher-block Chaining (CBC)

Key Exchange Algorithm

The KEA requires that each user have a private and public key, where the public key is available to both sender and receiver. There is no mention of how an end user is to validate the public keys of other users, however typical implementations use X.509 certificates. The private key of a user consists of 4 parameters:

p	1024-bit prime modulus
q	160-bit prime divisor of p-1 for public component checking
g	1024-bit base for the exponentiation, element of order q in the multiplicative group mod p.
x	160-bit user secret number ($0 < x < q$)

The modulus and other parameters are generated as per the Digital Signature Standard, FIPS-186 specification. The 1024-bit public key is then generated using the equation:

$$Y = g^{x} \bmod p$$

The KEA also requires the generation of 160-bit random number, which is then converted into a public number to be sent to the other end using the same equation as that for the public key:

$$R = g^{r} \bmod p$$

Users of a network share the same, p, q, and g values. For the KEA, this must be true, therefore only users of the same network can use the KEA.

Store and Forward (E-Mail) Applications

For store and forward applications there is no active connection between the two ends of communication. The sender generates all the data necessary for the KEA and then sends public versions of this information to the other end. The only data generated for store and forward applications is a single 160-bit private random number. This number is then converted into a public random number and sent to the receiver. For consistency in notation subscripts of A refer to the sender and subscripts of B refer to the receiver.

Sender Algorithm

The first step is to obtain the public key of the receiver and validate its authenticity. The next step is to verify the public key is from a user on the network, by checking that the following equations are satisfied:

$$1<Y_B < p \text{ and } (Y_B)^q \equiv 1 \bmod p$$

The sender generates a random 160-bit number and converts it to a public 1024-bit number using:

$$R_A = g^{r_A} \bmod p$$

Next the following values are computed:

$$t_{AB} = \left(Y_B\right)^{r_A} \bmod p = g^{r_A x_B} \bmod p$$

$$u_{AB} = \left(Y_B\right)^{x_A} \bmod p = g^{x_B x_A} \bmod p = g^{x_A x_B} \bmod p$$

$$w = \left(t_{AB} + u_{AB}\right) \bmod p$$

If, $w \neq 0$, then w is split into two different sections:

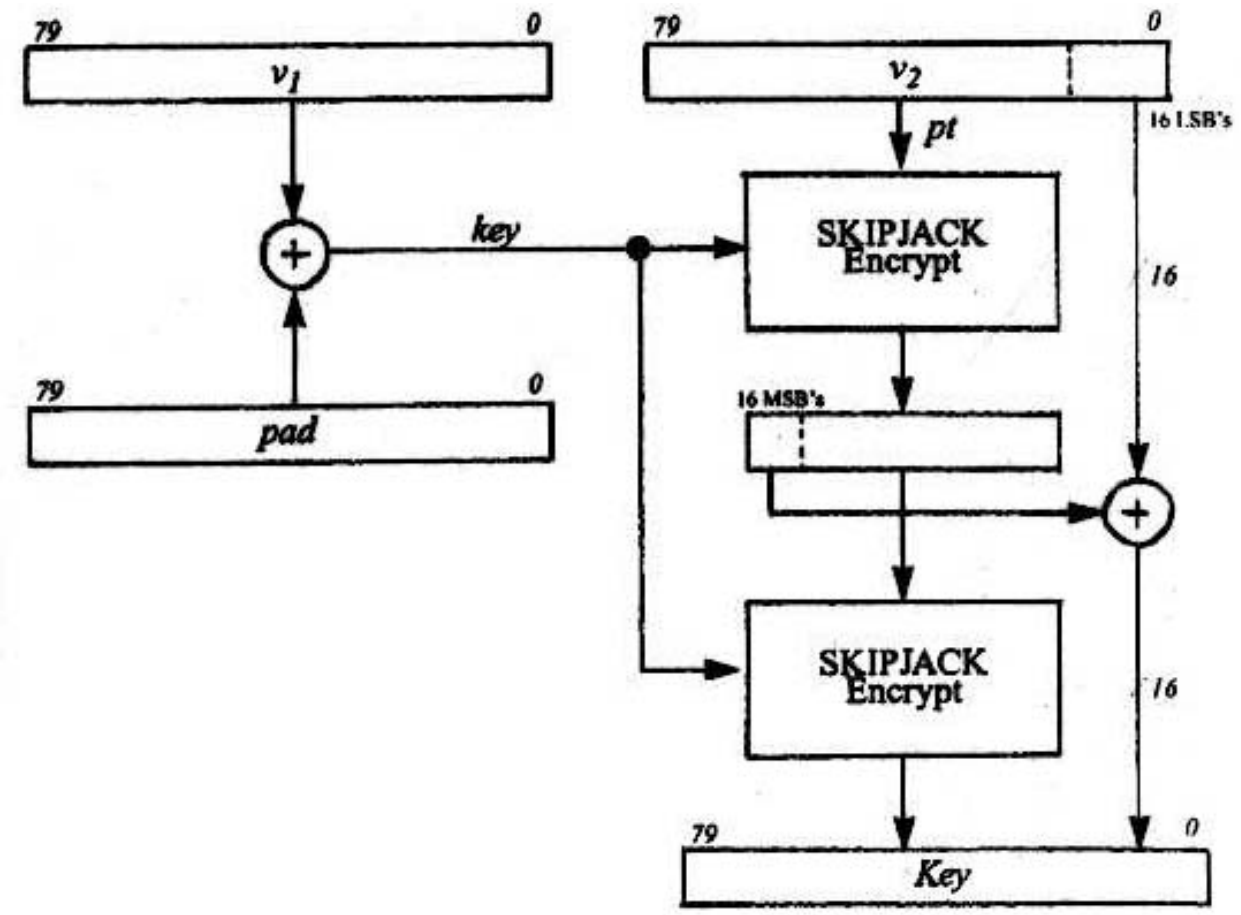
$$v_1 = \left(\frac{w}{2^{(1024-80)}}\right) \bmod 2^{80} \text{ and } v_2 = \left(\frac{w}{2^{(1024-160)}}\right) \bmod 2^{80}$$

Which is equivalent to saying that v_1 is the upper most 80 bits of w and v_2 is the penultimate upper 80 bits of w.

The key is then calculated using:

$$Key = 2^{16}\left[E_{v_1 \oplus pad}\left(E_{v_1 \oplus pad}\left[\frac{v_2}{2^{16}} \bmod 2^{64}\right]\right)\right] \oplus \left[\left(\frac{E_{v_1 \oplus pad}\left[\frac{v_2}{2^{16}} \bmod 2^{64}\right]}{2^{48}}\right) \oplus \left(v_2 \bmod 2^{16}\right)\right]$$

The pad used above is defined as 72f1a87e92824198ab0b hex. Schematically, this is represented by:



Receiver Algorithm

The first step is to obtain the public key of the sender and validate its authenticity as well as the public random number generated by the sender. The next step is to verify the public key and random number is from a user on the network, by checking that the following equations are satisfied:

$$1 < Y_A , R_A < p \text{ AND } (Y_A)^q \equiv 1 \bmod p \text{ and } (R_A)^q \equiv 1 \bmod p$$

Next the following values are computed:

$$t_{BA} = (R_A)^{X_B} \bmod p = g^{r_A x_B} \bmod p$$
$$u_{AB} = (Y_A)^{X_B} \bmod p = g^{x_A x_B} \bmod p$$

$$w = (t_{AB} + u_{AB}) \bmod p$$

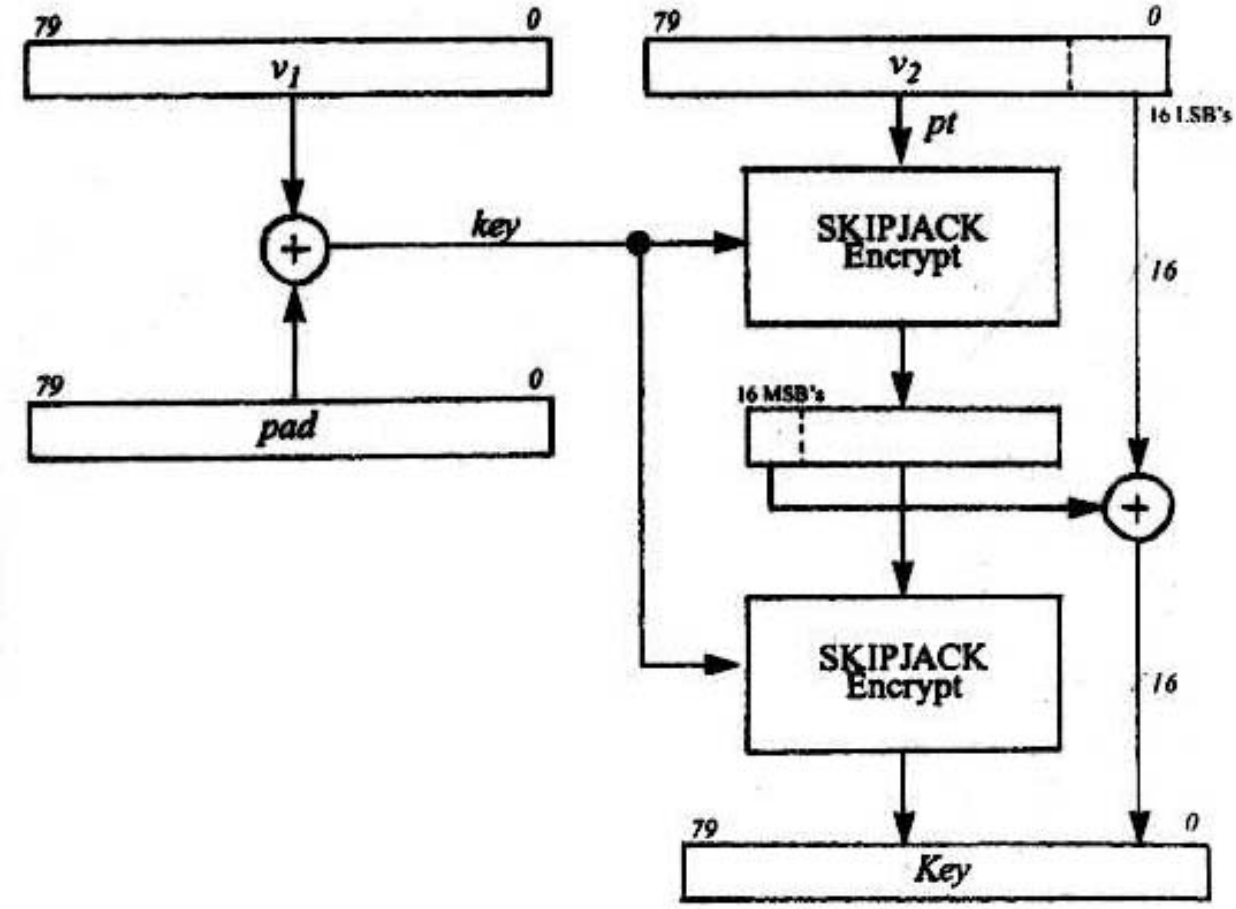
If, $w \neq 0$, then w is split into two different sections:

$$v_1 = \left(\frac{w}{2^{(1024-80)}} \right) \bmod 2^{80} \text{ and } v_2 = \left(\frac{w}{2^{(1024-160)}} \right) \bmod 2^{80}$$

Which is equivalent to saying that v_1 is the upper most 80 bits of w and v_2 is the penultimate upper 80 bits of w. They key is then calculated using:

$$Key = 2^{16}\left[E_{v_1 \oplus pad}\left(E_{v_1 \oplus pad}\left[\frac{v_2}{2^{16}} \bmod 2^{64}\right]\right)\right] \oplus \left[\left(\frac{E_{v_1 \oplus pad}\left[\frac{v_2}{2^{16}} \bmod 2^{64}\right]}{2^{48}}\right) \oplus \left(v_2 \bmod 2^{16}\right)\right]$$

The pad used above is defined as 72f1a87e92824198ab0b hex. Schematically, this is represented by:



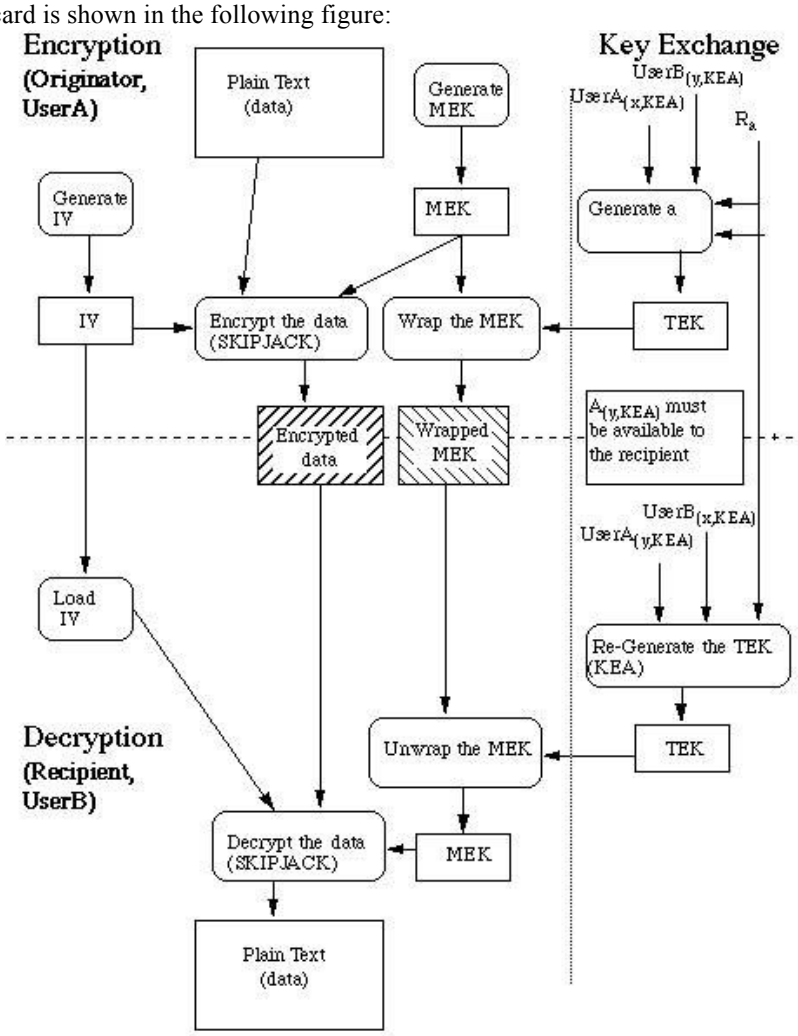
FORTEZZA KEA

The key generated by the KEA is not used directly to encrypt the information being sent the receiver. Rather a temporary message encryption key (MEK) is generated instead and used to encrypt the message. Then the key generated by the KEA, called a temporary encryption key (TEK) is used to encrypt the MEK. This encryption process is known as wrapping the MEK. There are no published details on the algorithm used to wrap the MEK. The implementation

used by me is to pad the MEK with six bytes of 06 (hexadecimal) to produce a 128-bit input. Which is then encrypted with SKIPJACK using the TEK for a key as two separate 64-bit blocks in the Codebook mode. What is important to notice is that to send a large file to multiple users it is only necessary to encrypt the large file once.

The FORTEZZA card uses the CBC mode for encryption, which requires a 64-bit initialization vector. Therefore, the information that needs to be sent to the receiver, along with the encrypted file, includes the wrapped MEK, the public random number, and the initialization vector. This is assuming that both parties have the public key of the other.

A summary of the KEA used to send a file or E-Mail message to another person using the FORTEZZA card is shown in the following figure:



SKIPJACK Optimization

This section discusses the optimizations made to the SKIPJACK implementation to increase the speed at which decryption and encryptions are performed. The optimizations were originally going to be made with assembly language routines. However, it was found that it was possible to optimize the code using the c programming language and the gcc compiler under Red Hat Linux 5.2.

Key Scheduling

The most noticeable optimization possible was to precompute combinations of each byte of the key with possible inputs to the F-table function. This is because the input to the F-table function is the exclusive or of any possible byte with a single byte of the key. Therefore, a table can be created which contains the 256 possible outputs of the F-table based on a single byte of the key. This will save four XOR instructions per step or 128 XOR instructions over a single encryption of a 64-bit block.

Memory Management

The stepping rules involve moving each 16-bit word to one place to the right each step. This is extremely slow and wasteful. It is possible to leave each 16-bit word in place and just change the operations around so that it is not necessary to perform 4 MOV instructions per step.Also, it was noticed that the Intel instruction set has only 4 general purpose registers EAX, EBX, ECX, and EDX, however penalties are incurred for using the ECX and EDX registers for mathematical operations. This does not leave enough registers to load all four 16-bit words into registers to optimize the speed at which functions can be performed upon the data. Another, subtle optimization, is one in which pointers are used to manipulate the data directly in memory and not to copy them into local variables.

Macros vs. Subroutines

The second biggest optimization comes from inlining the **G** and $\mathbf{G}^{-1}$ functions rather than subroutines. Subroutines require a lot of overheard to pass parameters and push the new instruction pointer on the stack. Therefore, by explicitly defining each step saves an enormous amount of time.

```
#define G(k0,k1,k2,k3,g1,g2,g5,g6) g5 = KeySchedTable[k0][g2]^g1; g6
= KeySchedTable[k1][g5]^g2; g5 = KeySchedTable[k2][g6]^g5; g6 =
KeySchedTable[k3][g5]^g6;

#define INV_G(k0,k1,k2,k3,g5,g6,g1,g2) g2 =
KeySchedTable[k3][g5]^g6; g1 = KeySchedTable[k2][g2]^g5; g2 =
KeySchedTable[k1][g1]^g2; g1 = KeySchedTable[k0][g2]^g1;
```

Compiler Optimizations

This is the most machine dependent part of the optimization process. The compiler used was gcc on Red Hat Linux 5.2. The processor used is a Pentium II 233 MHz machine. The command line to compile the optimized SKIPJACK routines is:

```
egcs -s -static -Wall -O3 -fomit-frame-pointer -funroll-loops -
malign-loops=2
-malign-functions=2 -malign-jumps=2 -mpentiumpro -o kea kea.c
optsjlib.c kealib.c -lcrypto
```

It is due to these compiler optimizations that the code generated is optimized greatly. Each round produces approximately 15 lines of assembly, which uses commands that do not suffer from penalties of using registers outside of the EAX and EBX.

Comparison with Optimized DES

For the optimized version of DES ,we used the SSLeay libraries implementation of DES. The SSLeay libraries use an extremely fast byte lookup implementation. To compare the two implementations a 64-bit test vector was used and two different keys, since DES requires a 56-bit key and SKIPJACK requires an 80-bit key. Both implementations

precompute key-scheduling values, therefore the timing is only for the encryption and decryption routines. In order to be able to get comparable values 100,000 encryptions and decryptions are performed and timed. The results are shown in the following table (average time to perform 100,000 iterations):

	Encryption	Decryption
SKIPJACK	34 ms	29 ms
DES	16 ms	16 ms

From a simple comparison standpoint, the DES algorithm consists of 16 rounds, whereas SKIPJACK is 32 rounds. It is conceivable then, that an optimized version of SKIPJACK would be twice as slow as that for DES. Using this simple comparison method, then the optimized version of SKIPJACK that we implemented is on target.

KEA Implementation

This covers the KEA implemented to transfer a file (or E-Mail) to another user on a network. It was desired to create a KEA that is compatible with the FORTEZZA PCMCIA security cards. However, since access to these cards was not available all the data that would normally be contained on the card is stored in text files. Also for some of the internal routines to the card, for which algorithms were not available from the FORTEZZA Application Developer's Guide, a best guess implementation was used. The KEA operates on numbers that are on the order of 1024 bits and therefore require a large number library to perform these functions. We used the SSLeay big number libraries for this purpose.

Public Key File

This is a simple text file that contains the 1024-bit public keys of users on the network. The format of the file is a unique identifier followed by an equal sign and then the users public key in hexadecimal format. Each record is kept on a separate line. This file is named public key.

```
ALICE=2D29ECD02E3497A67222D8DEBC286131D149F4581B3E586D0151024C02E8B23DA09A430
E2CA5ED1A4B2D772562316E4D2804D226788284ED655CF54610D38F66FAB1A0A2E2D3C6614401
901D9758D566722AFF1F734B2ADBD2B67F1300CE455F00968CA791A8767867363D7D49EE74A28
DC349D9FDFDB96B01F0FC1F0690EC96
BOB=7730D4BBF3A2EFDB218E70413E86102014CEC06C205F5419293B65C69A971E5455EB79A0B
DB90AB214C5240EDE6CFDD58C7C19C5269D57DFF60B61C1DB2FF64864BEE51987F270034BC390
AD731682095E42608C3D7987F9649FBF716887633EB574B39CC73DF89951FC1BD6D3889D48FE2
244B829AFD40506AB9221BA562C07
```

Private Key File

This is a text file containing the 4 components of a user's private key. The user's secret number (x), p, q, and g. Each component is on a separate line labeled by either X, P, Q, or G, an equal sign and the value in hexadecimal. This is stored in the file private.key.

```
X=62319AC47DE145180ABD322C59E2B6002781E494
P=9D4C6E6D42EA91C828D67D4994A9F01B8E5B5B730D0FAAE7BD569DD1914E3AD4759C805331E
DA1459FB56BE8A8DE4736652A82B276E82ACD63F5B78D0B75A03EB34D397DBE7B37408F72136A
CB0879FE61C718A37F5154B5078A7649FB3D4FB4C481E01062C5241F229FA580423368DD51090
DBF25351F0C5800DE05B92BA6A9
Q=97AD85FD2B371ED069818AB3C6EE8773D9DB029D
G=595D3443EC897C8251E5FA9D02AB8B75C0FC57B0969F880DA366A10001912A0196BCB81C41A
C8485031AC598B5481EAE2726B719D8D9915A6105973472386C0A6A2C732CD6700D341F54BF28
```

```
D12D692DE2FA05F55E898C2E20BB8A2602DB1BA07DE672E3B96D9AC29A18845063D918C32ED71
266B783311A0A8D08AC487BEA44
```

Key Exchange Data File

When a file is encrypted using the KEA program it is necessary to give the receiver the initialization vector, the public random number, and the wrapped MEK. Therefore a text file is created which contains these values in the same format as the public and private key file. The KEA data file has the same name as the output (encrypted) file with the extension .KEA-userid attached. Where userid is the unique identifier for the public key used in the KEA.

```
RA=4A3222E2767B90C49CF6EAD9AA634D6D2C74B2B513E14CE94B92E2259284E86E665B3FDCC4
45A79244ADE831987CC0AE38312F3E5171F06462C6D49FDE48AC3FAB7F395D8B752B40017E730
16480A06182F8444DF05BF412C92AD05603E1C0ECB86CC49366FFDE3717AB50BF57B6395CC1F8
4643EEE83F349C8F63CBF7DDBFC2
MEK=8F8787D92D6EA189EF1E2B28F04E83AB
IV=9B6FD1355C5DC91B
```

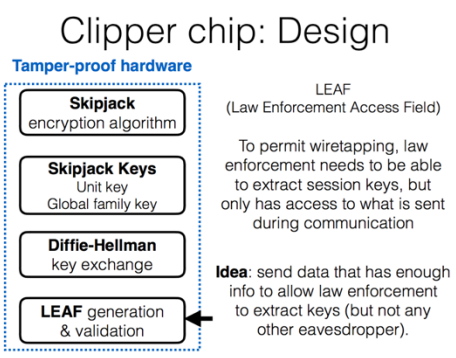
Conclusions

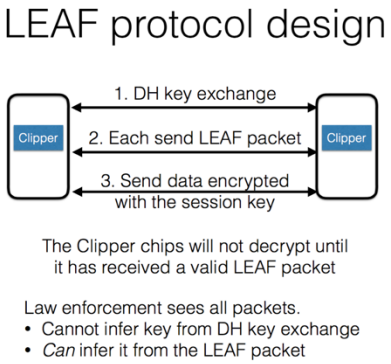
Creating an optimized SKIPJACK encryption and decryption implementation for financial transactions in digital currency was possible without using assembly code routines. The optimized versions are about twice as slow as those are for DES. For 100,000 encryptions the optimized SKIPJACK averaged 34 ms and 29 ms for decryption.

The KEA is mostly compatible with the FORTEZZA security cards. Where possible routines were implemented to mimic library calls that are available with the FORTEZZA Developer's Libraries. Only the wrapping of the MEK is not necessarily completely compatible since no information has been made available as to the algorithm for performing this function.

Skipjack Security

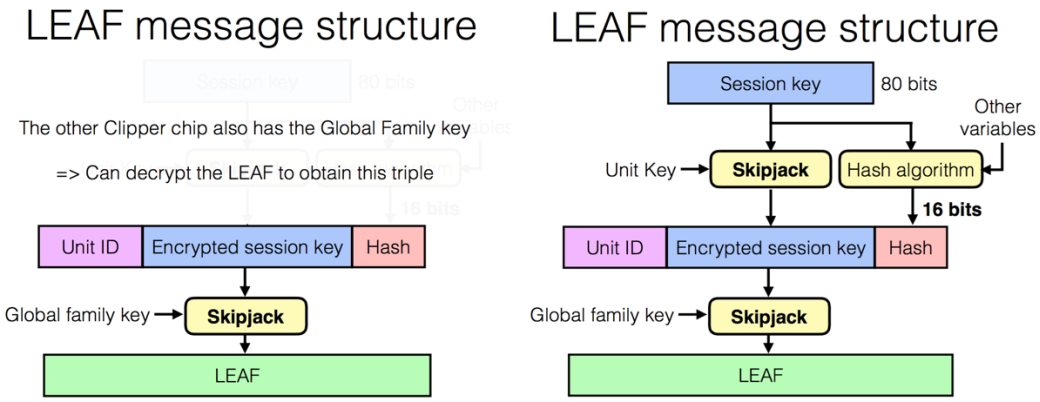
The objective of the SKIPJACK financial design was to provide a mechanism whereby persons could evaluate the strength of the classified encryption algorithm used in the escrowed encryption devices. Because SKIPJACK is but one component of a large, complex system, and because the security of communications encrypted with SKIPJACK depends on the security of the system as a whole, the design was extended to encompass other components of the system.





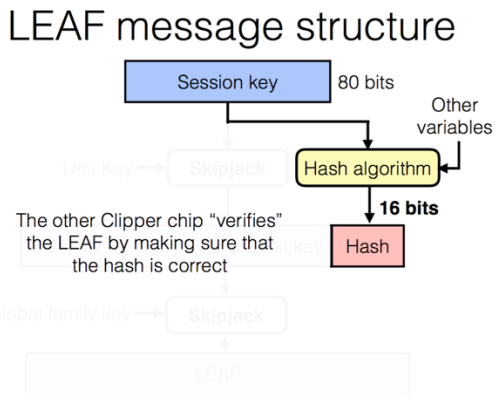
The results of our evaluation of the SKIPJACK algorithm are as follows:

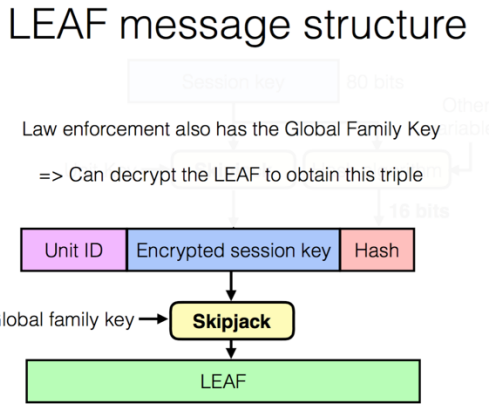
1 Under an assumption that the cost of processing power is halved every eighteen months, it will be 36 years before the cost of breaking SKIPJACK by exhaustive search will be equal to the cost of breaking DES today. Thus, there is no significant risk that SKIPJACK will be broken by exhaustive search in the next 30-40 years.
2 There is no significant risk that SKIPJACK can be broken through a shortcut method of attack.



1. A new technology initiative aimed at providing a high level of security for sensitive, unclassified financial transactions, while enabling lawfully authorized intercepts of telecommunications by law enforcement officials for criminal investigations. The initiative includes several components: A classified encryption/decryption algorithm called "SKIPJACK." Tamper-resistant cryptographic devices (e.g., electronic chips), each of which contains SKIPJACK, classified control software, a device identification number, a family key used by law enforcement, and a device unique key that unlocks the session key used to encrypt a particular communication. A secure facility for generating device unique keys and programming the devices with the classified algorithms, identifiers, and keys. Two escrow agents that each hold a component of every device unique key. When combined, those two components form the device unique key. A law enforcement access field (LEAF), which enables an authorized law enforcement official to recover the session key. The LEAF is created by a device at the start of an encrypted communication and contains the session key encrypted under the device unique key together with the device identifier, all encrypted under the family key. LEAF decoders that allow an authorized law enforcement official to extract the device identifier and encrypted session key from an intercepted LEAF. The identifier is then sent to the escrow agents, who return the components of the corresponding device unique key. Once obtained, the components are used to reconstruct the device unique key, which is then used to decrypt the session key. This reviews the security provided by the first component, namely the SKIPJACK algorithm. We investigated more than just SKIPJACK because the security of communications encrypted with the escrowed encryption technology depends on the security provided by all the components of the initiative, including protection of the keys stored on the devices, protection of the key components stored with the escrow agents, the security provided by the LEAF and LEAF decoder, protection of keys after they have been transmitted to law enforcement under court order, and the resistance of the devices to reverse engineering. In addition, the success of the technology initiative depends on factors besides security, for example, performance of the chips. Because some components of the escrowed encryption system, particularly the key escrow system, are still under design, we decided to issue this whitepaper on the security of the SKIPJACK.

2. Overview of the SKIPJACK Algorithm SKIPJACK is a 64-bit "electronic codebook" algorithm that transforms a 64-bit input block into a 64-bit output block. The transformation is parameterized by an 80-bit key, and involves performing 32 steps or iterations of a complex, nonlinear function. The algorithm can be used in any one of the four operating modes defined in FIPS 81 for use with the Data Encryption Standard (DES). The SKIPJACK algorithm was developed by NSA and is classified SECRET. It is representative of a family of encryption algorithms developed in 1980 as part of the NSA suite of "Type I" algorithms, suitable for protecting all levels of classified data. The specific algorithm, SKIPJACK, is intended to be used with sensitive but unclassified information. The strength of any encryption algorithm depends on its ability to withstand an attack aimed at determining either the key or the unencrypted ("plaintext") communications. There are basically two types of attack, brute-force and shortcut.





3. Susceptibility to Brute Force Attack by Exhaustive Search In a brute-force attack (also called "exhaustive search"), the adversary essentially tries all possible keys until one is found that decrypts the intercepted communications into a known or meaningful plaintext message. The resources required to perform an exhaustive search depend on the length of the keys, since the number of possible keys is directly related to key length. In particular, a key of length N bits has 2^N possibilities. SKIPJACK uses 80-bit keys, which means there are 2^{80} (approximately 10^{24}) or more than 1 trillion trillion possible keys. An implementation of SKIPJACK optimized for a single processor on the 8-processor Cray YMP performs about 89,000 encryptions per second. At that rate, it would take more than 400 billion years to try all keys. Assuming the use of all 8 processors and aggressive vectorization, the time would be reduced to about a billion years. A more speculative attack using a future, hypothetical, massively parallel machine with 100,000 RISC processors, each of which was capable of 100,000 encryptions per second, would still take about 4 million years. The cost of such a machine might be on the order of $50 million. In an even more speculative attack, a special purpose machine might be built using 1.2 billion $1 chips with a 1 GHz clock. If the algorithm could be pipelined so that one encryption step were performed per clock cycle, then the $1.2 billion machine could exhaust the key space in 1 year. Another way of looking at the problem is by comparing a brute force attack on SKIPJACK with one on DES, which uses 56-bit keys. Given that no one has demonstrated a capability for breaking DES, DES offers a reasonable benchmark. Since SKIPJACK keys are 24 bits longer than DES keys, there are 2^{24} times more possibilities. Assuming that the cost of processing power is halved every eighteen months, then it will not be for another 24 * 1.5 = 36 years before the cost of breaking SKIPJACK is equal to the cost of breaking DES today. Given the lack of demonstrated capability for breaking DES, and the expectation that the situation will continue for at least several more years, one can reasonably expect that SKIPJACK will not be broken within the next 30-40 years.

Conclusion 1: Under an assumption that the cost of processing power is halved every eighteen months, it will be 36 years before the cost of breaking SKIPJACK by exhaustive search will be equal to the cost of breaking DES today. Thus, there is no significant risk that SKIPJACK will be broken by exhaustive search in the next 30-40 years.

4. Susceptibility to Shortcut Attacks In a shortcut attack, the adversary exploits some property of the encryption algorithm that enables the key or plaintext to be determined in much less time than by exhaustive search. For example, the RSA public-key encryption method is attacked by factoring a public value that is the product of two secret primes into its primes. Most shortcut attacks use probabilistic or statistical methods that exploit a structural weakness, unintentional or intentional (i.e., a "trapdoor"), in the encryption algorithm. In order to determine whether such attacks are possible, it is necessary to thoroughly examine the structure of the algorithm and its statistical properties.

4.1 Design and Evaluation Process SKIPJACK was designed using building blocks and techniques that date back more than forty years. Many of the techniques are related to work that was evaluated by some of the world's most accomplished and famous experts in combinatorics and abstract algebra. SKIPJACK's more immediate heritage dates to around 1980, and its initial design to 1987. SKIPJACK was designed to be evaluatable, and the design and evaluation approach was the same used with algorithms that protect the country's most sensitive classified information. The specific structures included in SKIPJACK have a long evaluation history, and the cryptographic properties of those structures had many prior years of intense study before the formal process began in 1987. Thus, an arsenal of tools and data was available. This arsenal was used by dozens of adversarial evaluators whose job was to break SKIPJACK. Many spent at least a full year working on the algorithm. Besides highly experienced evaluators, SKIPJACK was subjected to cryptanalysis by less experienced evaluators who were untainted by past approaches. All known methods of attacks were explored, including differential cryptanalysis. The goal was a design that did not allow a shortcut attack. The design underwent a sequence of iterations based on feedback from the evaluation process. These iterations eliminated properties which, even though they might not allow successful attack, were related to properties that could be indicative of vulnerabilities. "I believe that SKIPJACK can only be broken by brute force there is no better way." In summary, SKIPJACK is based on some of best technology. Considerable care went into its design and evaluation in accordance with the care given to algorithms that protect classified data.

4.2 Independent Analysis and Testing Our own analysis and testing increased our confidence in the strength of SKIPJACK and its resistance to attack.

4.2.1 Randomness and Correlation Tests: A strong encryption algorithm will behave like a random function of the key and plaintext so that it is impossible to determine any of the key bits or plaintext bits from the ciphertext bits (except by exhaustive search). We ran two sets of tests aimed at determining whether SKIPJACK is a good pseudo random number generator. These tests were run on a Cray YMP. The results showed that SKIPJACK behaves like a random function and that ciphertext bits are not correlated with either key bits or plaintext bits.

4.2.2 Differential Cryptanalysis : Differential cryptanalysis is a powerful method of attack that exploits structural properties in an encryption algorithm. The method involves analyzing the structure of the algorithm in order to determine the effect of particular differences in plaintext pairs on the differences of their corresponding ciphertext pairs, where the differences are represented by the exclusive-or of the pair. If it is possible to exploit these differential effects in order to determine a key in less time than with exhaustive search, an encryption algorithm is said to be susceptible to differential cryptanalysis. However, an actual attack using differential cryptanalysis may require substantially more chosen plaintext than can be practically acquired. We examined the internal structure of SKIPJACK

to determine its susceptibility to differential cryptanalysis. We concluded it was not possible to perform an attack based on differential cryptanalysis in less time than with exhaustive search.

4.2.3 Weak Key Test Some algorithms have "weak keys" that might permit a shortcut solution. DES has a few weak keys, which follow from a pattern of symmetry in the algorithm. We saw no pattern of symmetry in the SKIPJACK algorithm which could lead to weak keys. We also experimentally tested the all "0" key (all 80 bits are "0") and the all "1" key to see if they were weak and found they were not.

4.2.4 Symmetry Under Complementation Test The DES satisfies the property that for a given plaintext-ciphertext pair and associated key, encryption of the one's complement of the plaintext with the one's complement of the key yields the one's complement of the ciphertext. This "complementation property" shortens an attack by exhaustive search by a factor of two since half the keys can be tested by computing complements in lieu of performing a more costly encryption. We tested SKIPJACK for this property and found that it did not hold.

4.2.5 Comparison with Classified Algorithms We compared the structure of SKIPJACK to that of NSA Type I algorithms used in current and near-future devices designed to protect classified data. This analysis was conducted with the close assistance of the cryptographer who developed SKIPJACK and included an in-depth discussion of design rationale for all of the algorithms involved. Based on this comparative, structural analysis of SKIPJACK against these other algorithms, and a detailed discussion of the similarities and differences between these algorithms, our confidence in the basic soundness of SKIPJACK was further increased.

Conclusion 2: There is no significant risk that SKIPJACK can be broken through a shortcut method of attack.

5. Secrecy of the Algorithm The SKIPJACK algorithm is sensitive for several reasons. Disclosure of the algorithm would permit the construction of devices that fail to properly implement the LEAF, while still interoperating with legitimate SKIPJACK devices. Such devices would provide high quality cryptographic security without preserving the law enforcement access capability that distinguishes this cryptographic initiative. Additionally, the SKIPJACK algorithm is classified SECRET NOT RELEASABLE TO FOREIGN NATIONALS. This classification reflects the high quality of the algorithm, i.e., it incorporates design techniques that are representative of algorithms used to protect classified information. Disclosure of the algorithm would permit analysis that could result in discovery of these classified design techniques, and this would be detrimental to national security. However, while full exposure of the internal details of SKIPJACK would jeopardize law enforcement and national security objectives, it would not jeopardize the security of encrypted communications. This is because a shortcut attack is not feasible even with full knowledge of the algorithm. Indeed, our analysis of the susceptibility of SKIPJACK to a brute force or shortcut attack was based on the assumption that the algorithm was known.

Conclusion 3: While the internal structure of SKIPJACK must be classified in order to protect law enforcement and national security objectives, the strength of SKIPJACK against a cryptanalytic attack does not depend on the secrecy of the algorithm.

BREAKING DOWN 'Skipjack Virtual Money Supply'

The skipjack encrypcurrency or virtual money supply reflects the different types of liquidity each type of virtual money or digital currency has in the economy. It is broken up into different categories of liquidity or spendability.

We analyze the money supply and develop virtual money policies revolving around it through controlling interest rates and increasing or decreasing the amount of skipjack encrypcurrency money flowing in the digital economy. Skipjack encrypcurrency supply data is collected, recorded and published periodically in the Fiestel Core Network (FCN), typically by FCN as skipjack central bank.

The encrypcurrency supply is also known as the encrypcurrency stock.

Plural	skipjack
Symbol	
Ticker symbol	SJK
Subunits	
$^{1}/_{10}$	micro Jack (mJ) / dimes
Skipjack	Unspent outputs of transactions (in multiples of a dimes)
Development	
Original author(s)	Mr. SkipJack
White paper	SkipJack and the World's First Encrypcurrency
Reference implementation	SkipJack FCN Core
Initial release	0.1.0 / 14 January 2018
Latest release	
Website	Skipjackx.com
Ledger	
Ledger start	20 January 2018
Timestamping scheme	Message Authentication Codes (MAC)
Protocol	CBC-MAC, HMAC Authentication Encryption CCM,EAX,GCM and OCB
Central Bank	Feistel Core Network (FCN)
Issuance	Block Cipher
Block reward	NA
Block time	NA
Block explorer	superencryp.org (mid 2019)
Circulating supply	
Supply limit	100,000,000,000 (Year 2118)
Valuation	

How Encrypcurrency Supply is Measured

The various types of encrypcurrency in our virtual money supply are generally **classified as Es, such as E0, E1, E2 and E3** according to the type and size of the Encryp account in which the instrument is kept. Not all of the classifications are widely used, and it may use different classifications.

E0 and E1, for example, are also called narrow virtual encrypmoney and include dimes that are in circulation and other encrypcurrency equivalents that can be converted easily to cash. E2 includes E1 and, in addition, short-term time deposits in skipjack merchants as encrypbanks and certain encrypcurrency market funds. E3 includes E2 in addition to long-term deposits.

The FCN measures and publishes the total amount of E1 and E2 encrypcurrency supplies on daily basis. They can be found online in the platform.

The Effect of Encrypcurrency Supply on the Economy

An increase in the supply of encrypcurrency typically lowers virtual currency p2p loan interest rates, which in turn, generates more investment and puts more virtual money in the hands of consumers, thereby stimulating spending. The increased business activity raises the demand. The opposite can occur if the money supply falls or when its growth rate declines.

Monetary exchange equation

The digital currency supply is important because it is linked to inflation by the equation of exchange in an equation proposed by Irving Fisher in 1911:

$$E*V = P*Q$$

where

- ***E*** is the total digital currency in the worldwide circulation's encrypcurrency supply,
- ***V*** is the number of times per year each encrypcurrency is spent (velocity of virtual money),
- ***P*** is the average price of all the goods and services sold during the year,
- ***Q*** is the quantity of assets, goods and services sold during the year.

In mathematical terms, this equation is an identity which is true by definition rather than describing economic behavior. That is, velocity is defined by the values of the other three variables. Unlike the other terms, the velocity of encrypcurrency has no independent measure and can only be estimated by dividing PQ by E. Some adherents of the quantity theory of encrypmoney assume that the velocity of encrypcurrency is stable and predictable, being determined mostly by financial institutions. If that assumption is valid then changes in E can be used to predict changes in PQ. If not, then a model of V is required in order for the equation of exchange to be useful as a macroeconomics model or as a predictor of prices.

We replace the equation of exchange with equations for the demand for virtual money which describe more regular and predictable economic behavior. However, predictability (or the lack thereof) of the velocity of virtual money is equivalent to predictability (or the lack thereof) of the demand for encrypmoney (since in equilibrium encrypcurrency demand is simply Q/V).

In practice, we almost always use real GDP to define Q, omitting the role of all transactions except for those involving newly produced goods and services (i.e., consumption goods, investment goods, government-purchased goods, and exports). But the original quantity theory of encrypmoney did not follow this practice: PQ was the monetary value of all new skipjack transactions, whether of real goods and services or of virtual assets.

Rates of growth

In terms of percentage changes (to a close approximation, under low growth rates), the percentage change in a product, say XY, is equal to the sum of the percentage changes %ΔX + %ΔY). So, denoting all percentage changes as per unit of time,

%ΔP + %ΔQ = %ΔE + %ΔV.

This equation rearranged gives the basic inflation identity:

%ΔP = %ΔE + %ΔV – %ΔQ.

Inflation (%ΔP) is equal to the rate of money growth (%ΔE), plus the change in velocity (%ΔV), minus the rate of output growth (%ΔQ). So if in the long run the growth rate of velocity and the growth rate of real GDP are exogenous constants (the former being dictated by changes in payment institutions and the latter dictated by the growth in the economy's productive capacity), then the monetary growth rate and the inflation rate differ from each other by a fixed constant.

As before, this equation is only useful if %ΔV follows regular behavior. It also loses usefulness if the central bank lacks control over %ΔE.

What is the 'Matching Trade Multiplier Effect'

The matching trade multiplier effect is the expansion of a skipjack encrypcurrency supply that results from merchant being able to lend. The size of the matching trade multiplier effect depends on the percentage of deposits that merchant are required to hold as reserves. In other words, it is the encrypcurrency used to create more currency and is calculated by dividing total merchant deposits by the reserve requirement.

1. Deposit Multiplier
2. Reservable Deposit
3. Reserve Requirement
4. Deposit Broker

BREAKING DOWN 'Matching Trade Multiplier Effect'

The matching trade multiplier effect is basically the amount of dimes that merchant generate with each dimes of reserves, or the ratio of deposits compared to reserves that are circulating in the encrypcurrency system, means increasing the amount of dimes in the encryp supply by taking in deposits, keeping some in reserves (an amount prescribed by the FCN) and lending out the rest.

The term can also be referred to as the encryp multiplier.

Visualizing the Matching Trade Multiplier Effect

To calculate the effect of the matching trade multiplier effect on the encrypcurrency supply, start with the amount merchant initially take in through deposits, and divide this by the reserve ratio. If, for example, the reserve requirement is 20 percent, for every 100 dimes a customer buy the encrypcurrency and deposits into a merchant, 20 dimes must be kept in reserve. However, the remaining 80 dimes can be loaned out to other merchant customers. This 80 dimes is then deposited

by these customers into another merchant, which in turn must also keep 20 percent, or 16 dimes, in reserve but can lend out the remaining 64 dimes.

This cycle continues as more people buy and deposit encrypcurrency and more merchant continue lending it until finally the 100 dimes initially deposited creates a total of 500 dimes (100/0.2) in deposits. This creation of dime deposits is the matching trade multiplier effect.

Required Dime Reserves

The dime reserve required is set by the board of the Fiestel Core Network (FCN) and it varies based on the total amount of liabilities held by a particular depository merchant. For merchant dime deposits are required to hold 10 percent of their total liabilities in reserve as a start.

Skipjack Dime Supply and the Multiplier Effect

The encrypcurrency supply consists of multiple levels. The first level, referred to as the monetary base, refers to all of the dimes in circulation within an economy. The next two levels, E1 and E2 add the balances of deposit accounts and those associated with small-denomination time deposits and retail dime market shares, respectively.

As a customer makes a deposit into an E1 deposit account, the merchant can lend the funds beyond the reserve to another person. While the original depositor maintains ownership of the initial deposit, the funds created through lending are generated based on those funds. If the borrower subsequently deposits the funds received from the lending merchant, this raises the value of E1 even though no additional physical currency actually exists to support the new amount.

The higher the reserve requirement, the tighter the encrypcurrency supply, which results in a lower multiplier effect for every dime deposited. This may make merchant less inclined to lend as their options to do so are limited based on the size of the reserve. In contrast, the lower the reserve requirement, the larger the encrypcurrency supply which means more encrypmoney is being created for every dime deposited, and merchant may be more inclined to take additional risks with the larger pool of available funds.

Definition

The encryp matching trade is defined in various ways. Most simply, it can be defined either as the statistic of "merchant encrypcurrency"/"FCN currency", based on the actual observed quantities of various empirical measures of encrypcurrency supply, such as E2 (broad money) over E0 (base encrypcurrency), or it can be the theoretical "maximum commercial bank money/central bank money" ratio, defined as the reciprocal of the reserve ratio, 1/RR.The matching in the first (statistic) sense fluctuates continuously based on changes in merchant encrypcurrency and FCN (though it is *at most* the theoretical multiplier), while the matching trade in the second (legal) sense depends only on the reserve ratio, and thus does not change unless the law changes.

For purposes of monetary policy, what is of most interest is the *predicted impact* of changes in FCN encrypcurrency on merchant encrypcurrency, and in various models of monetary creation, the associated multiple (the ratio of these two changes) is called the matching trade (associated to that model). For example, if one assumes that people hold a constant fraction of deposits as dime, one may add a "currency drain" variable (currency–deposit ratio), and obtain a multiplier of (1+CD) / (RR+CD)

These concepts are not generally distinguished by different names; if one wishes to distinguish them, one may gloss them by names such as **empirical** (or **observed**) multiplier, **legal** (or **theoretical**) multiplier, or **model** multiplier, but these are not standard usages.

Similarly, one may distinguish the *observed* reserve–deposit ratio from the legal (minimum) reserve ratio, and the *observed* currency–deposit ratio from an assumed model one. Note that in this case the reserve–deposit ratio and currency–deposit ratio are *outputs* of observations, and fluctuate over time. If one then uses these observed ratios as model parameters (*inputs*) for the predictions of effects of monetary policy and assumes that they remain constant, computing a constant multiplier, the resulting predictions are valid only if these ratios do not in fact change. Sometimes this holds, and sometimes it does not; for example, increases in FCN may result in increases in merchant encrypcurrency – and will, if these ratios (and thus matching) stay constant – or may result in increases in excess reserves but little or no change in merchant encrypcurrency, in which case the reserve–deposit ratio will grow and the matching trade will fall.

Mechanism

There are two suggested mechanisms for how encrypcurrency supply creation occurs in a ***fractional-encrypreserve*** **(FER)** system: either reserves are first injected by the FCN, and then lent on by the merchant, or loans are first extended by merchants, and then backed by reserves borrowed from the FCN.

Reserves first model

In the "reserves first" model of encrypcurrency creation, a given reserve is lent out by a merchant, then deposited at a merchant (possibly different), which is then lent out again, the process repeating and the ultimate result being a geometric series.

Formula

The matching trade multiplier, *m*, is the inverse of the reserve requirement, *RR*:

$$m = \frac{1}{RR}$$

This formula stems from the fact that the sum of the "amount loaned out" column above can be expressed mathematically as a geometric series with a common ratio of 1- RR

To correct for currency drain (a lessening of the impact of monetary policy due to peoples' desire to hold some currency in the form of cash) and for banks' desire to hold reserves in excess of the required amount, the formula:

$$m = \frac{(1 + CurrencyDrainRatio)}{(CurrencyDrainRatio + DesiredReserveRatio)}$$

can be used, where "Currency Drain Ratio" is the ratio of cash to deposits, i.e. C/D, and the Desired Reserve Ratio is the sum of the Required Reserve Ratio and the Excess Reserve Ratio.

The formula above is derived from the following procedure. Let the monetary base be normalized to unity. Define the legal reserve ratio, $\alpha \in (0,1)$, the excess reserves ratio, $\beta \in (0,1)$, the currency drain ratio with respect to deposits, $\gamma \in (0,1)$; suppose the demand for funds is unlimited; then the theoretical superior limit for deposits is defined by the following series:

$$Deposits = \sum_{n=0}^{\infty} [(1 - \alpha - \beta - \gamma)]^n = \frac{1}{\alpha + \beta + \gamma}$$

Analogously, the theoretical superior limit for the encrypcurrency held by public is defined by the following series:

$$PubliclyHeldCurrency = \gamma \cdot Deposits = \frac{\gamma}{\alpha + \beta + \gamma}$$

and the theoretical superior limit for the total loans lent in the market is defined by the following series:

$$Loans = (1 - \alpha - \beta) \cdot Deposits = \frac{1 - \alpha - \beta}{\alpha + \beta + \gamma}$$

By summing up the two quantities, the theoretical matching trade multiplier is defined as

$$m = \frac{MoneyStock}{MonetaryBase} = \frac{Deposits + PubliclyHeldCurrency}{MonetaryBase} = \frac{1 + \gamma}{\alpha + \beta + \gamma}$$

where $\alpha + \beta = DesiredReserveRatio$ and $\gamma = CurrencyDrainRatio$

The process described above by the geometric series can be represented in the following table, where

- loans at stage k are a function of the deposits at the precedent stage:

$$L_k = (1 - \alpha - \beta) \cdot D_{k-1}$$

- publicly held encrypmoney at stage k is a function of the deposits at the precedent stage:

$$PHM_k = \gamma \cdot D_{k-1}$$

- deposits at stage k are the difference between additional loans and publicly held money relative to the same stage:

$$D_k = L_k - PHM_k$$

Process of encrypcurrency multiplication:

n	Deposits	Loans	Publicly Held Money
$n = 0$	$D_0 = 1$	-	-
$n = 1$	$D_1 = (1 - \alpha - \beta - \gamma)$	$L_1 = (1 - \alpha - \beta)$	$PHM_1 = \gamma$
$n = 2$	$D_2 = (1 - \alpha - \beta - \gamma)^2$	$L_2 = (1 - \alpha - \beta)(1 - \alpha - \beta - \gamma)$	$PHM_2 = \gamma(1 - \alpha - \beta - \gamma)$
$n = 3$	$D_3 = (1 - \alpha - \beta - \gamma)^3$	$L_3 = (1 - \alpha - \beta)(1 - \alpha - \beta - \gamma)^2$	$PHM_3 = \gamma(1 - \alpha - \beta - \gamma)^2$
...	...	...	...
$n = k$	$D_k = (1 - \alpha - \beta - \gamma)^k$	$L_k = (1 - \alpha - \beta)(1 - \alpha - \beta - \gamma)^{k-1}$	$PHM_k = \gamma(1 - \alpha - \beta - \gamma)^{k-1}$
...	...	...	...
$n \to \infty$	$D_\infty = 0$	$L_\infty = 0$	$PHM_\infty = 0$
	Total Deposits:	**Total Loans:**	**Total Publicly Held Money:**
	$D = \frac{1}{\alpha + \beta + \gamma}$	$L = \frac{1 - \alpha - \beta}{\alpha + \beta + \gamma}$	$PHM = \frac{\gamma}{\alpha + \beta + \gamma}$

Table

This re-lending process (with no currency drain) can be depicted as follows, assuming a 20% reserve ratio and a 100 dime initial deposit:

Individual Merchant	Amount Deposited	Merchant Lent Out	FCN Reserves
A	100.00	80.00	20.00
B	80.00	64.00	16.00
C	64.00	51.20	12.80
D	51.20	40.96	10.24
E	40.96	32.77	8.19
F	32.77	26.21	6.55
G	26.21	20.97	5.24
H	20.97	16.78	4.19
I	16.78	13.42	3.36
J	13.42	10.74	2.68
K	10.74		

		Total Reserves:
		89.26
Total Amount of Deposits:	Total Amount Lent Out:	Total Reserves + Last Amount Deposited:
457.05	357.05	100.00

Example

For example, with the reserve ratio of 20 percent, this reserve ratio, *RR*, can also be expressed as a fraction:

$$RR = \frac{1}{5}$$

So then the matching trade multiplier, *m*, will be calculated as:

$$m = 1/\frac{1}{5} = 5$$

This number is multiplied by the initial deposit to show the maximum amount of money it can be expanded to.

Another way to look at the monetary multiplier is derived from the concept of encrypmoney supply and encrypmoney base. It is the number of dime supply that can be created for every dime of monetary base. EncrypMoney supply, denoted by M, is the stock of money held by public. It is measured by the amount of currency and deposits. EncrypMoney Base, denoted by B, is the summation of currency and reserves. Currency and Reserves are monetary policy that can be affected by the FCN. For example, the FCN can increase currency by circulating more dimes and they can similarly increase reserve by requiring a higher percentage of deposits to be stored in the FCN.

Mathematically: Let M= C+D and B =C+R where

M=EncrypMoney Supply
C=Currency
D=Deposits
B=EncrypMoney Base
R=Reserve

By algebraic manipulation

$$\frac{M}{B} = \frac{C+D}{C+R} = \frac{C+D}{C+R}.\frac{D/(CR)}{D/(CR)}$$

$$M = B.\frac{C+D}{C+R}.\frac{D/(CR)}{D/(CR)}$$

$$= B.\frac{\frac{D}{R}(1+\frac{D}{C})}{\frac{D}{R}+\frac{D}{C}}$$

is the $\frac{\frac{D}{R}(1+\frac{D}{C})}{\frac{D}{R}+\frac{D}{C}}$ multiplier. Therefore, if encrypmoney base is held constant, the ratio of D/R and D/C affects the encrypcurrency supply. When the ratio of deposits to reserves (D/R) reduces, the multiplier reduces. Similarly, if the ratio of deposits to currency (D/C) falls, the multiplier falls as well.

The multiplier effect is relevant to considering monetary and fiscal policies, as well how the encrypcurrency system works. For example, the deposit, the monetary amount a customer deposits at a merchant, is used by the merchant to loan out to others, thereby generating the encrypmoney supply. Merchant are required to reserve a certain ratio of the customer's deposits in reserve, either in the form of encrypwallet vault or of a deposit maintained by the FCN. Therefore, if the FCN (and hence its monetary policy) requires a higher percentage of reserve, then it lowers the merchant 's financial ability to loan.

Loans first model

In the alternative model of encrypcurrency creation, loans are first extended by merchant – say, 1,000 dime of loans (following the example above), which may then require that the merchant borrow 100 dimes of reserves either from depositors (or other private sources of financing), or from the FCN.

Implications for monetary policy

The matching trade multiplier plays a key role in skipjack monetary policy, and the distinction between the multiplier being the *maximum* amount of merchant encrypcurrency created by a given unit of FCN encrypcurrency and approximately *equal* to the amount created has important implications in monetary policy.

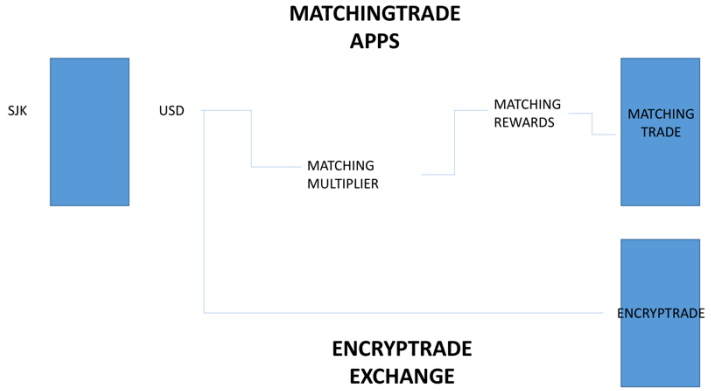
Liquidity at Encryptrade

Skipjack Encryptrade is built to provide liquidity to the non-liquid Encrypcurrency economy. Skipjack is designing the encryptrade Platform, a single globally-sourced trading platform (Encryptrade) with an associated suite of services (Prime Brokerage) which will enable the highest level of liquidity. In the process, this will allow anybody to tap into all the opportunities the new encryp economy has to offer. The Encryptrade Platform will be managed and operated by Skipjack Corporation established under laws of Singapore.The Encryptrade is designed based on the three fundamentals and application:

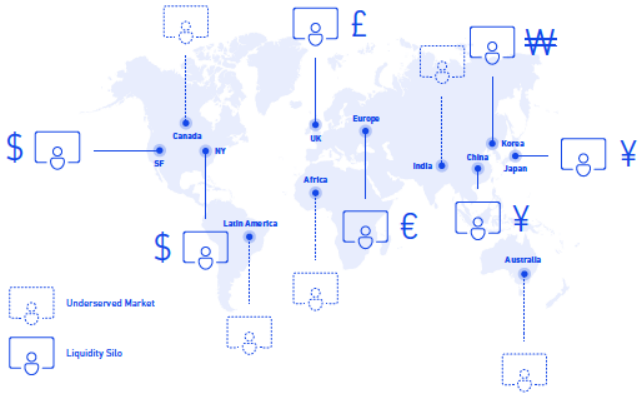
- **DERIVATIVE TRADING**
- **BINARY OPTION TRADING**
- **MATCHINGTRADE APPS**

The Encryptrade:

- offers a mobile wallet, offline storage, and insurance protection for currency stored on servers.
- Supports several fiat currencies: British Pounds, US Dollars, Euros, Canadian Dollars, Australian Dollars and Singapore Dollars.
- Facilitates instant buy-sell orders for USD/SJK pair with withdrawals and deposits available in currencies other than dollars as well. The exchange offers trading using limit orders where a pre-determined price can be set for buy and sell orders.
- To buy skipjack, clients need to open an account with encryptrade, which is followed by transferring money into the account. This can be done through SEPA, international wire transfer, etc. For those using SEPA, there is an additional step for change over from Euros to Dollars for trading and then Dollars to Euro while withdrawal. There is a fee levied on withdrawals - the fee for SEPA is fixed at 0.90 after conversion into Euros while the fee for any international withdrawal is 0.09% (minimum fee being $15). Encryptrade earns a trading fee on the successful trades which is based on the 30 day trading history of clients. The minimum fee rate is as low as 0.5% - usually for new accounts and those with a thin volume (<$500). (Fee Schedule)

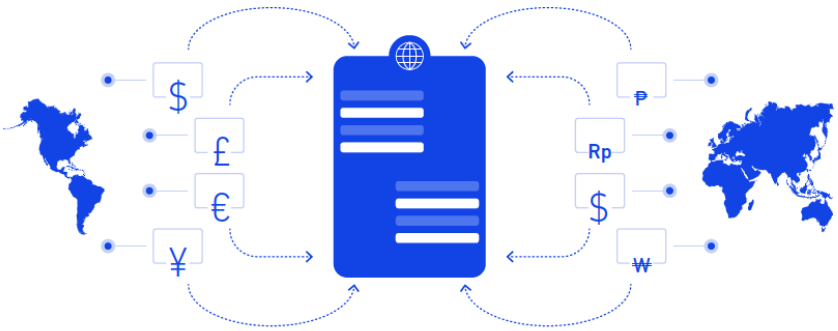


In a sense, the various encrypcurrency exchanges are Liquidity Silos. They are liquid in some pairs, but their liquidity is only accessible to clients of that exchange. And while most of the exchanges offer liquidity in certain highly liquid currencies (USD, EUR, CNY, JPY, etc.), there is no convenient and highly-liquid on-ramp into the Encryp world for holders of minor currencies such as CAD, NZD, SGD, PHP, IDR, etc. Separately, each market's liquidity remains small, but collectively they represent a large untapped source of liquidity that now desperately wants an on-ramp into the Encrypworld.



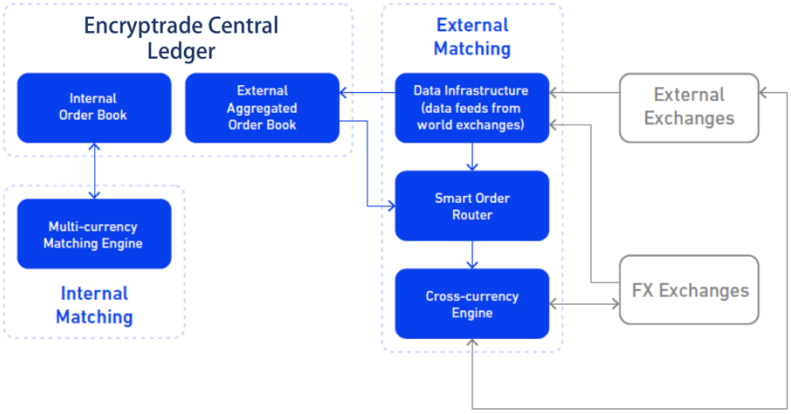
The Encryptrade Platform brings together the entire global network of encrypcurrency exchanges and makes them accessible to everybody. With access to all the major reputable exchanges worldwide, the Encryptrade Platform provides an unparalleled and powerful suite of trading services to Individual and Institutional investors alike—as well as to Dime Holders.

Encryptrade
Multi-Market Order Central Ledger (CL)
The Encryptrade Platform's Central Ledger offers a multi-market order book that aggregates the [orders /prices] on world's various liquidity sources into a single highly liquid tradable order book, allowing orders to be placed in the currency of one's choice:



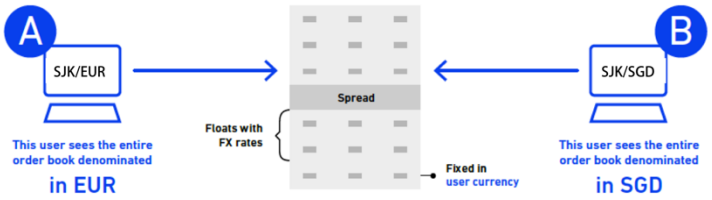
The Central Ledger solves two important problems that limit liquidity in the Encrypcurrency world: Liquidity Silos and Underserved Markets. As mentioned above, Liquidity Silos are pockets of liquidity that exist because liquidity at one exchange in one currency pair (e.g. SJK/USD at exchange A) cannot easily access the opposite liquidity if it exists on another exchange (e.g. SJK/EUR at exchange B). Underserved Markets are smaller markets that, in isolation, are not yet big enough to support their own liquid Encryp Dime. People in the countries that only have Liquidity Silos and Underserved Markets would like to be able to trade the Encryp Dime in their home currencies, but there is no highly-liquid on-ramp into the Encryp world for them. The Liquidity Silos and Underserved Markets problems are solved by connecting these exchanges into a single point of liquidity, and by allowing the order book to be priced in any of the major currencies and many minor currencies.

Internal Order Book: An order book that contains FX-adjusted orders for all orders placed by users of the Central Ledger. External Aggregated Order Book: An order book that contains all other orders (but FX adjusted) that exist throughout the world other than those placed in Internal Order Book. Each order in this book is linked to an order on the various exchanges internationally.



The Encryptrade Platform's Central Ledger supports trading in the quote currency of one's choice. To understand how useful this is, it is helpful to understand that trading pairs in the Encrypcurrency and FX worlds are composed of a Base Currency (what you are buying and selling, e.g.: SJK, etc.), and a Quote Currency (the currency in which you see the price, e.g.: USD, EUR, etc.). It is also possible to have pure Skipjack Dime pairs such as SJK/BTC – which has a Base Currency of SJK, and Quote Currency of BTC. Traders using the Central Ledger have the option of selecting their Quote Currency from any of the major fiat currencies, plus some Encryp Skipjack Dimes.

For example, a German trader (A) wishing to sell BTC (Base Currency) can choose to see the Central Ledger for BTC in EUR (their Quote Currency). When they place an order in the Central Ledger, they see their order enter the SJK/EUR market, and the order's price is reflected in their orderbook in EUR. While, a Singapore trader, using the Central Ledger,will see the entire order book in their own currency of choice, likely SGD. They will see the order that was placed in SJK/EUR by the German Trader, priced in SGD – and from the Singapore trader's perspective, the German trader's order will fluctuate with changes in the EUR/SGD rates. If the order to sell SJK/EUR is matched with an order to buy SJK/SGD, an FX conversion happens behind the scenes. Both parties receive their execution in the respective currencies of their orders.



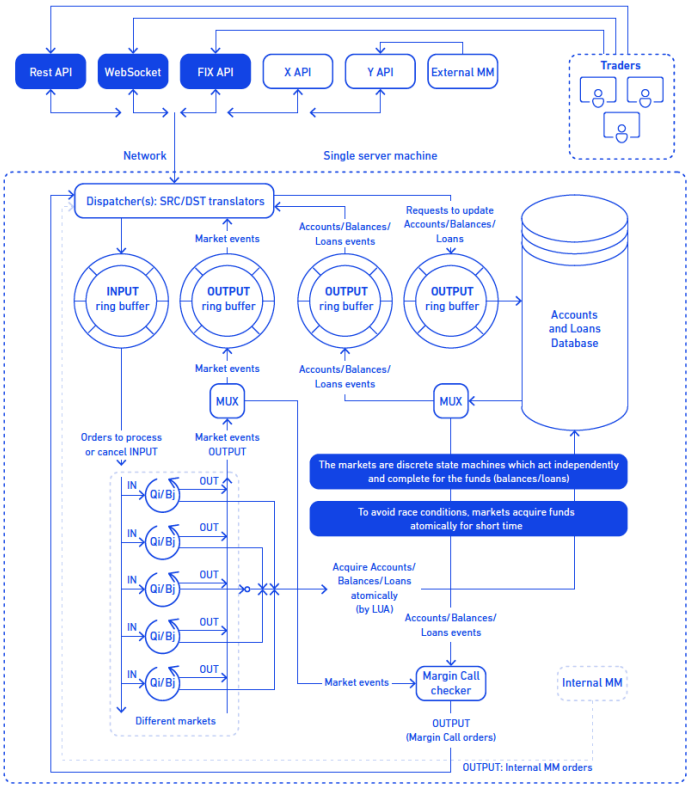
The Architecture

Underpinning Encryptrade Platform's Central Ledger are three technologies that are already used extensively by Skipjack on a daily basis:

1. Matching Engine (ME)
2. Cross Currency Conversion Engine (CCCE)
3. Smart Order Routing (SOR)

Matching Engine (ME):

Built from scratch by combining decades of experience in financial technology, the Central Ledger Matching Engine (ME) is capable of processing several million transactions per second, making it one of the most advanced matching engines in the industry. The ME has been architected to be hyper-scalable to support a very high number of markets (Skipjack Dime pairs) and offers native support for a number of key order types, around which a powerful Order Management System has been built. All these features, coupled with "built to fail" resiliency, make the Central Ledger's ME one of the best in the market.



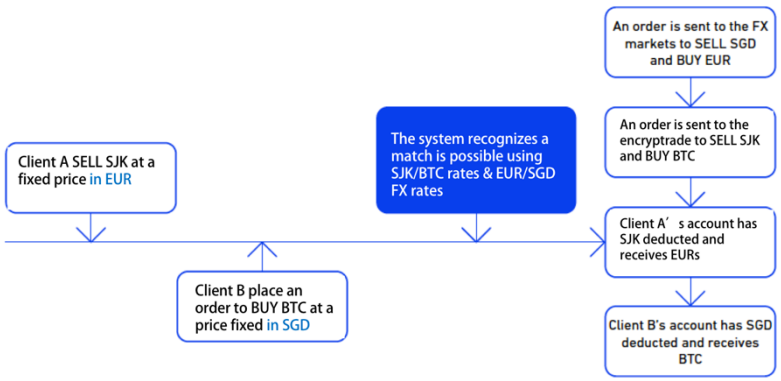
Qi, Q - queue of requests, i - queue identifier

Bj, B - order book, j - order book identifier

SRC - source, DST - destination, MUX - multiplexer

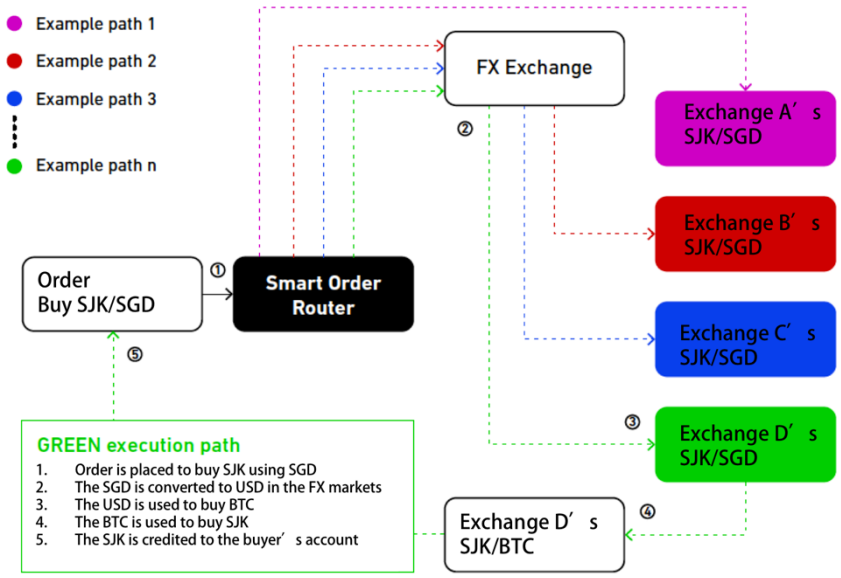
Cross Currency Conversion Engine (CCCE)

The Cross Currency Conversion Engine allows for near-instant, automated currency conversions: When an execution is confirmed, and the two parties don't share the same Quote Currency, a Quote Currency conversion is necessary. For example, if two clients of the Central Ledger are both trading SJK (the Base Currency), but one of them is trading it in SGD and the other in USD, an FX conversion using USD/SGD must be added to the transaction to complete the match. The CCCE automatically performs this when the match is confirmed. It is also constantly adjusting, in real-time pricing, the order book to reflect the current executable exchange rates. It should be mentioned here that this currency conversion is not limited to fiat. For example, using SJK/BTC for currency conversion, it becomes possible to provide a match between an order placed in SJK/SGD, and one placed in BTC/SGD. Furthermore, adding a fiat FX conversion allows for matches between pairs such as SJK/EUR and BTC/SGD (see diagram below. Using the CCCE allows the Central Ledger and Smart Order Routing (SOR) to source liquidity from multiple places that are not normally matched, allowing the Central Ledger to tap into additional sources of liquidity.



Smart Order Routing (SOR):

The Smart Order Routing Technology maintains low-latency, real-time (R/T) feeds for all major exchanges throughout the world. When any of those order books changes, it's reflected in Encryptrade Platform's Central Ledger. When an order is placed on the Central Ledger, and no match is available internally, the Smart Order Routing Technology checks to see if there is a match available on another exchange. If so, it automatically routes the order in the currency of the order on the other exchange.



Prime Brokerage

The Encryptrade Platform's Prime Brokerage is comprised by a suite of services that provide users with tools to reduce counterparty risk and increase ROI. Prime Brokerage includes so-called Direct Market Access, Fiat Management, and Credit Facility as its core offering. Users are now able to trade directly on any global reputable exchange, without even having an account or funds on those exchanges.

Encryptrade Platform users will gain the following benefits:

1. Reduction of Counterparty Risk: Users of the Encryptrade Platform will only ever need to deal with Skipjack as their counterparty.
2. Increase in Capital Efficiency: Encryptrade Platform users will be able to deal on other exchanges without having to move funds directly. Capital movements will be done via Skipjack's funding facilities, and should be instant for both Encryp Dime and fiat.
3. Netting of positions: Users of the Encryptrade Platform will have the ability to net positions taken across different trading venues. This will allow market participants to better take advantage of inefficiencies in the sometimes fractured crypto markets.

Direct Market Access
The Prime Brokerage allows its users to leverage Skipjack's Smart Order Routing (SOR) technology and connectivity to all reputable global exchanges, and get Direct Market Access for the fastest and best execution from a single platform. With our state-of-the-art trading dashboards, users of the Encryptrade Platform will be able to manage their trading books, place orders, monitor risk and keep an eye on all global markets on major exchanges, 24/7.

Our Order Management System (OMS) and Matching Engine (ME) provide unparalleled banking-level performance capable of processing millions of orders per second. Our OMS provides a professional-grade FIX API, with both common and advanced order types and execution algorithms.

Fiat Management
Fiat Management is one of the most difficult (and restricted) constraints when it comes to providing liquidity to the market. Moving large sums of fiat is slow, expensive and time consuming – which is one of the reasons Skipjack Dime are the future of finance. Skipjack built an extensive network of relationships with banks. Part of our banking strategy has been to co-locate our bank accounts with those of other major exchanges, by creating accounts in the same bank—and in many cases the same branch—as other exchanges.

We have also begun working with a major global bank to provide fiat transfer optimization. This will allow us to complete immediate fiat transfers globally. This service will also be available to third party liquidity providers using the Prime Brokerage. Skipjack is planning to obtain a Funds Transfer Service Provider License and any other permits or licenses when required for Liquidity. There are many third party liquidity providers that provide only limited liquidity. Their liquidity provision is substantially constrained by the time and cost of moving fiat. By providing this service as part of our Prime Brokerage, third party liquidity providers will be incentivized to use the platform—because it will allow them to better monetize their capital.

Crypto/Fiat Credit
The Encryptrade Platform's Prime Brokerage will extend its Skipjack/Fiat Credit facility to users to leverage their existing balances for enhanced trading opportunities. Skipjack/ Fiat Credit facility allows users to borrow using either fiat or Skipjack Dime as collateral. Skipjack is also in the process of applying for a formal Banking License, to expand the range of services we can offer our users. Skipjack will also obtain any other permits or licenses when required.

Real Time Reporting
The Prime Brokerage users will have access to customized activity statements, to view detailed information about their account activity including positions, cash balances, transactions, and more. They will be able to run trade confirmation reports to view all executions.

Other Services
System Co-Location
Our Encryptrade Platform provides co-location on one of our powerful private servers physically located near our Matching Engine and OMS system—or near liquidity venues within our network of exchange partners. We provide support for mainstream programming languages. Users of the Encryptrade Platform will be able to deploy their own strategies and also offer them to other customers on the platform at a fee.

Automated Trading Strategies

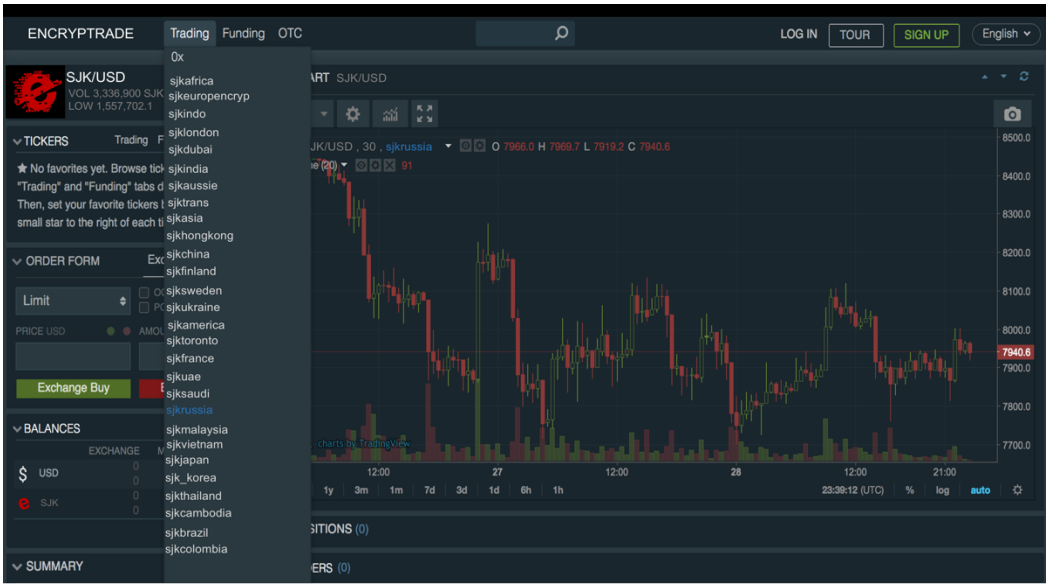
Our Encryptrade Platform provides access to third party trading strategies, and to its own automated trading and market-making strategies. Users can select from a published list of automated algorithmic strategies, then have their asset traded using these strategies. Strategies will be published and will be accessible by our users. Historical performance and other details on strategies will be available to users to aid them in the selection process. Initially, these strategies will be provided by Skipjack. However the platform, data, and tools we use will be made available to thousands of data scientists throughout the industry, allowing access to any individual or teams with the talent/skill to design and offer their own trading strategies.

Examples of trading strategy use cases:

1. Use automated market making strategies to do market making for their dimes.
2. Talented and trading system developers can design their own trading systems and then use them to trade their own assets—or share them publicly for a share of the profits and/or a fee.
3. Dime holders can select one or a basket of strategies and then have their asset automatically traded using the automate strategies. Automated Trading Strategies will be provided only to the extent permitted by the applicable laws, regulations and rules.

Trading Tools

Users of the Encryptrade Platform have access to best-in-class trading tools developed over the years for Skipjack. Among these tools, users have full access to a powerful trading dashboard, developed for Skipjack and adapted for the Encryptrade Platform. Users have access to tools to manage their trades and positions, monitor global markets in real time, enter and manage orders, view live charts, etc.



Skipjack is a virtual currency and digital asset designed, to work as a currency. It is commonly referred to with terms like digital currency, digital cash, virtual currency, electronic currency, or encrypcurrency. The question whether skipjack is a currency is a potential. Skipjack have three useful qualities in a currency, they are "hard to earn, limited in supply and easy to verify". Economists define money as a store of value, a medium of exchange, and a unit of account and agree that skipjack has some way to go to meet all these criteria. It does best as a medium of exchange.

Payment service providers

Merchants accepting skipjack ordinarily use the services of skipjack payment service providers. When a customer pays in skipjack, the payment service provider accepts the skipjack on behalf of the merchant, converts it to the local currency, and sends the obtained amount to merchant's bank account, charging a fee for the service.

As an investment and Fund of Funds (FoF)

Some encrypmoney lovers will buy skipjack to protect their savings against high inflation or as future investment referring to crypto currency bitcoin success. Other methods of investment are skipjack fund of funds. The first regulated skipjack fund of funds (FoF) will establish in Quarter 4 ,2018 and according to market forecast in 2019 there will be 1,000,000 encryp wallets with more than $1 billion worth of skipjack. The exact number of skipjack millionaires is will be produce is still uncertain as every encrypwallet have different value.

Superencryp Block is the world's new breakthrough leading software platform for digital assets.

Offering the most security production block chiper algorithm in the world, we are using new technology to build a radically better, secure and fast financial system. Skipjack software will revolutionize and empowered users in across the globe to transact more security, quickly and without costly. We also offer real time transaction data for users to analyze the digital economy. Skipjack latest new encrypcurrency technology platform is based on block cipher algorithms that ensure the most efficient bid-offer matching across natural peer-to-peer flow as well as third party -exchanges. All historical transactions are stored in a dedicated data-warehouses that continuously analyses the data to enhance risk management and identify predictive behaviors, and in turn enable Skipjack to optimize the encrypcurrency exchange process and better educate the customer on possible payment strategies, for example, it may highlight different payment mixes depending on the current encrypcurrency valuations. In other words, everything possible is done to ensure that the customer gets the best deal.

As the first real digital currency-called as encrypcurrency, the woven thread in Skipjack is one of evolution. It first appeared in January 2018, to be trade as a type of digital karma between members. Skipjack is a new encrypcurrency: no one knew what it was worth. So an exchange rate of 10 to 1 was created, linking Skipjack to the US Dollar. With a set value, Skipjack design to be use inside Feistel Core Network (FCN) worldwide, and could be traded between members on a global basis for free.

To mitigate exchange rate risks the currency was designed to be diversified to a basket of currencies (making it highly stable) and the decision was taken to back issuance 100% with assets, in accordance with the algorithmic index that formulated the Skipjack. Soon this basket included supply and demand , commodities like gold and silver, and finally, carbon, making Skipjack the first environmentally linked currency in the world.

With a high level of diversified stability and conservative structure, Skipjack will show promise as a global currency. But it will the introduction of carbon to the underlying that give Skipjack social DNA, embedding environmental support in every transaction as a derivative benefit. The more Skipjack in the world, the more carbon purchased for asset reserves, and the more world is protected.

Skipjack is rapidly scaling into the global financial system, becoming the first digital encrypcurrency used for commodity trades, plan to be the first traded in regulated FX markets, and the first to enter bond markets and national bourses. It will be the most stable currency in the world, 50% less volatile than traditional fiat currencies. With no leverage and no interest, it ticks the box for some aspects of Islamic finance. These attributes make Skipjack a yawn for speculators and cowboys, but a boon for producers and common people.

Skipjack is issued and regulated by Fiestel Core Network, a centralized superencryp block work as federal reserve with Artificial Intelligence (AI) embedded by end 2019 and the Skipjack Central Reserve Board, a group of financial experts from within the community tasked with protecting the integrity of Skipjack. Skipjack and Feistel Core Network (FCN) are encased in a secure Trust with the sole task of protecting the technological assets and intellectual property that governs the community.

Verified Nodes

The Skipjack network assumes nodes can be trusted, since any node can join the network at any time and validate by a FCN and matching block. In contrast, payment platforms such as credit card networks rely on a single trusted party to clear transactions, and require trust of all nodes. Skipjack targets a different approach, where distributed nodes are run and control by verified transactions which the system design to use Fiestel Core Network (FCN) with a collective incentive to maintain the network.

The Skipjack protocol is designed to create and maintain an evenly centralized network. The FCN will verify the identity of the initial nodes to help bootstrap the network across a broad geographic distribution. The network will initially accept approximately a few hundred nodes, targeting across 150+ nations. The FCN can then expand to more networks based on transparent nomination and acceptance (for example by 80% of verified nodes) until most geographies have several operational nodes.

"Encrypcurrency " means that the currency does not have physical representation and its distribution, flow and exchange is recorded on the Internet, with policy managed by the Skipjack FCN Central Authority. This is a markedly different strategy than the hundreds of decentralized crypto currencies that place their faith in a fixed supply horizon that is not governed by a central point taking responsibility for its actions. While decentralized currencies have their merit, the human input in Skipjack is designed to guide flexibility for changing conditions and to ensure the viability of the currency in changing circumstances.

Governing rules regarding Skipjack provide simple fixed conditions: an inbuilt social contract with endemic support for the environment and fixed issuance relative to asset reserves to promote stability, reliability and security. Together these attributes support the 4 Core Attributes of Skipjack: stability, globality, security and support for nature.

Skipjack Authorities

In order to make Skipjack more accessible, Feistel Core Network (FCN) authorises independent entities known as "Authorities" to manage liquidity of Skipjack currency. Organizations may acquire this status for various purposes: a bank would typically operate accounts on behalf of their customers. A currency exchange would be doing realtime currency trading. A NGO might issue relief aid or microfinance support. A corporation might convert assets to Skipjack to hedge their balance sheet or to meet carbon obligations. An investment fund might use authority status to hold and control large amount of Skipjack for a stable long term investment solution.

Regular users and merchants who only have one account and do not operate large volumes of Skipjack do not require Authority status. Authority status comes with a legal responsibility to make sure that all Skipjack activities are legal and ethical. Through those principles Feistel Core Network (FCN) ensures Skipjack availability to anyone through any means while remaining a legal, stable, 100% backed asset acknowledged and respected worldwide.

Skipjack Issuance

Skipjack is held in the FCN, an offline ‘cold’ encrypvault. When purchased, Skipjack ‘encrypted’ from the FCN into circulation, and the corresponding purchase value in fiat currency is held in reserves allocated to the underlying basis components of Skipjack. Skipjack is a currency backed by a collection of other currencies, commodities and carbon credits. Technically this means for every Skipjack in a member account there are corresponding frozen assets held in reserve. This ensures the stability of Skipjack prices but also means that all Skipjack must be accounted for and backed in the Central Skipjack Reserve.

Skipjack is distributed through Authorities. If you access an Authority to buy Skipjack for $100, this amount is broken down and invested into dollars, euros, pounds, yen, yuan and many other currencies, commodities and carbon credits. When you sell Skipjack the underlying assets are released. This process is called Reserve Balancing. The process is managed through a series of sophisticated hedging algorithms that balance the underlying reserves and issuance in real time with live financial markets data, updating as frequently as several times per second.

Skipjack Pricing API

Skipjack pricing is updated in realtime and slightly fluctuates depending on underlying performance. Feistel Core Network (FCN) makes Skipjack prices publicly available to anyone through an API, however this free public API is only updated once per hour.As an Authority, entities have access to high frequency pricing which offers real time pricing with precision of more than 0.1 second, to ensure that Authorities have a very accurate pricing before during and after purchase or sale. Skipjack purchase / sales orders can also be distributed through FIX protocol, which is popular among financial institutions worldwide.

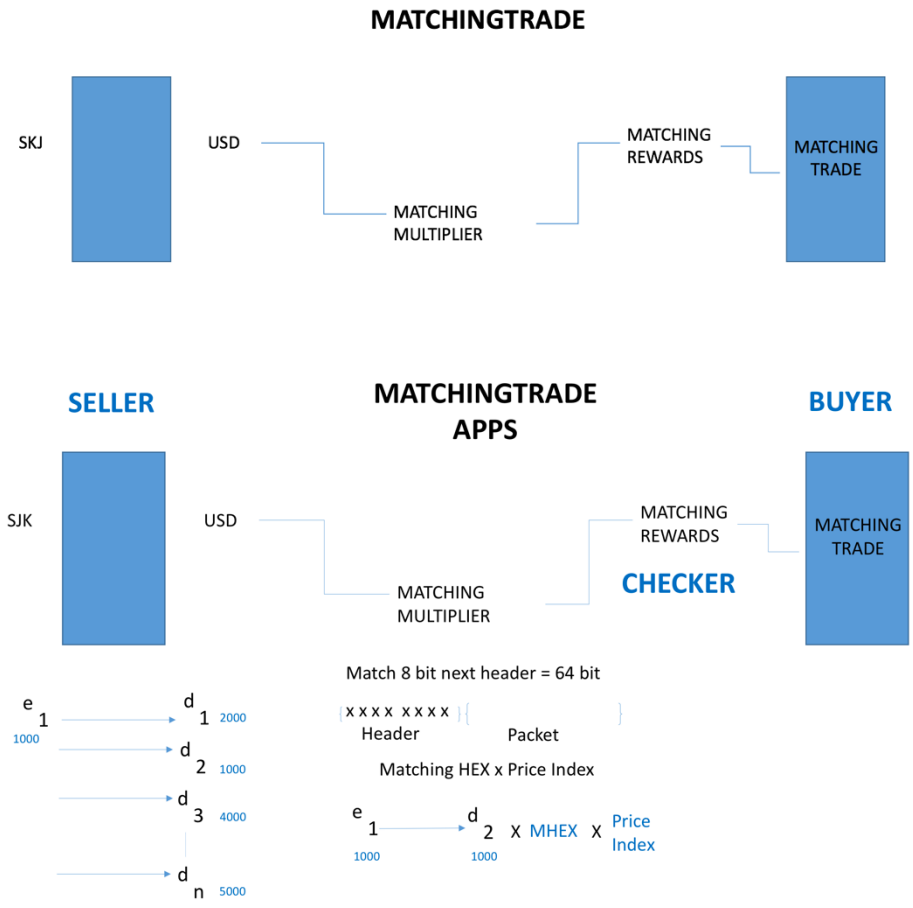
Efficiency

Skipjack Matching Trade is a new design platform to process the validation of each transaction by the public and a very intensive process by the public checker and FCN due to the design of the protocol. Skipjack's worker-lottery approach to distributing new dimes was designed to make it economically costly for any single participant to take control of the network. Skipjack Matching Trade have verified transaction distribute block rewards to other transactions and users. Instead of a transaction keeping all of the block reward, the majority will be immediately distributed to all other verified transactions and qualified users, to remove any financial incentive to game the system. Since the network will grow in a gradual manner to only elected, verified organizations, network hash rate can be kept consistent across cooperating nodes. This will enable Skipjack maintaining integrity of the system, since computational effort creates collective benefit across the network.

Incentives "Checker " Rewards

The Skipjack network operates based on collective incentive and individual reward. When a matching block is successfully verified by "checker", transactions are confirmed to all other nodes immediately. The financial motivation to increase the efficiency of transaction is increased, since reward is distributed to all other checker. The motivation for the node operator becomes to provide just enough efficiency to verify that transactions are valid within a collectively agreed upon level of maximum latency. Thus the platform becomes collectively motivated to achieve optimal transaction efficiency.

Each node will have visibility into all other nodes using a superencryp block-explorer and system stats accessible at superencrypblock.com/node/name, and lazy nodes that do not meet the agreed upon service requirements can be removed from the pool of nodes eligible for receiving rewards. Additionally, since all nodes have aligned incentives the protocol will encourage cooperation and collaboration when making decisions.



Usability

Usability is critical to the adoption of a new technology platform. Skipjack seeks to set a new standard in terms of comprehension, brand perception and interaction design to provide the best possible user experience. Many cryptocurrencies developed to date have been primarily technology projects, written by developers for developers, and have not prioritized the usability and understandability of their systems. The perceived complexity and risk of using most platforms limits mainstream adoption, and simplicity is not a core value of most projects.

Skipjack aims to reach the simplicity and ubiquity of a system such as mPesa within Kenya, but with global availability. Skipjack will prioritize simplicity and ease-of-use in all design decisions, to help accelerate platform adoption and network effects. Skipjack will enable users to create verified accounts, and profiles such as skipjackx.com/alice or skipjackx.com/bob will be formed to enable usage such as "send 3 skipjack to bob" via interfaces like texting or voice command. The Skipjack Foundation plans to invest in community projects and applications, which will be available at skipjackx.com/apps. Grants will initially be overseen by a Platform Committee appointed by the Foundation board of directors.

Scalability

Skipjack will start with larger blocksizes and shorter blocktimes, enabled by a fast network of servers. Targeting an initial confirmation time of less than 1 minute, Skipjack will enable fast transactions with low fees. Our goal is an initial transactional capacity of approximately 1,000,000 transactions per second, facilitating a faster payment network for use-cases like remittance and wire transfers. Within a few years, Skipjack aims to achieve capacity of 10,000,000+ transactions per second by using Transaction Sharding, where transactions are directed to specific shards for acceptance and validation, instead of the whole network validating the same transaction. Transaction sharding will combine principles of centralization, identity attestation, trust and reputation within a system of verified nodes that establish and maintain the network.

Dime Allocation

Beyond creating a resilient open financial system, Skipjack seeks to reduce the inequality that currently exists in the distribution of digital currency. Skipjack plans to algorithmically allocate dimes to its community over several years, ensuring that value created is distributed fairly over time.

Skipjack plans to distribute 50% of the dime supply to the first 1 billion unique, verified human users on the platform (with equitable demographic and geographic representation) to allocate value created to a large community of users. 20% will be allocated to verified nodes and their network of researchers and developers. 10% will be held by the Skipjack Foundation to fund operations and community grants, and 10% to active contributors and advisors. The remaining 10% will be allocated to strategic partners worldwide.

If properly executed, this dime allocation strategy will accelerate platform adoption and network effects, and help achieve a gradual reduction of economic inequality across the globe. Skipjack will give the majority of its value created to billions of users around the world. This can help more than two billion people in developing nations gain improved access to financial services, and achieve a more balanced distribution of resources.

Monetary Policy

Skipjack aims to become a medium-of-exchange as well as a store-of-value, with sufficient dime supply to provide good system usability. Instead of fixing monetary supply to an arbitrary level, Skipjack aims to increase the dime supply gradually in proportion to its market capitalization, to help maintain reasonable dime prices by increasing supply incrementally. In order to enable billions of users to eventually own many Skipjack dime each, an initial supply of 1 billion dimes will be generated over several years. This will enable a future user to have hundreds of Skipjack in their account, instead of a very small fraction of a more scarce dime. For example, future user would find it much easier to pay 1.25-skipjack for something than 0.000000125-btc.

One Skipjack dime will be divisible into 100 parts, to mimic the familiar concepts of dollars and cents. This will increase overall system usability, by reducing the cognitive load of dealing with very small fractions. Further research is needed into the best mechanism for algorithmic distribution of new dimes, but in principle half of dimes created should be given to unique, verified human users over a period of several years.

If new dimes are issued in the future beyond the supply of 1 trillion, Skipjack seeks to maintain the policy of distributing 50% to end-users, in a fair algorithmic manner. With an algorithm designed to minimize volatility, Skipjack should eventually reach sufficient price stability to function as a daily-use currency. This would eventually enable Skipjack to function as both a medium-of-exchange and a store-of-value.

Governance

Technology will improve over time, and Skipjack systems will be upgraded as they scale. Skipjack will collaborate with a broad group of experts to ensure continuous improvements are made in a timely manner based on community feedback. Initially, a diverse board of directors will design Version 1.0 of the protocol, and invite a network of verified nodes to run the software. New directors may be approved by majority vote of existing directors, and will rotate every few years based on expertise needed during the next phase of growth.

The Skipjack Foundation will initially use off-chain governance processes currently employed by large public organizations, to ensure a team of experts can iterate on system design with public oversight. The directors will set up special committees of expert contributors, with the most knowledgeable foundation director as committee chair. These committee chairs will coordinate collaboration in the Skipjack Forum, for design planning and policy discussions. Independent audit committees will be elected by the foundation directors, who will have full transparency into the systems of all network nodes, and report all findings to the public.

Skipjack will take an empirical approach to governance, iterating over time to become more centralized and more robust as system adoption increases. As Skipjack becomes more popular, governance processes will migrate into the protocol itself, using weighted-reputation voting mechanisms to achieve full system self- governance within a few years. Progressive centralization will allow new ideas to be tested and verified before being widely distributed, and will allow Skipjack to gradually achieve the topology of a global mesh network.

The Skipjack Foundation

The Skipjack Foundation is an independent, non-profit organization creating an open-source, centralized financial system. The mission of the Skipjack Foundation is to develop a global currency protocol that is evenly distributed and extremely robust. Skipjack was created with the belief that effective digital currencies have the potential to give greater financial control and independence to billions of people. The Skipjack foundation will guide the development of the Skipjack protocol, with transparent governance and continual improvement.

Skipjack is in the early stages of system design, and aims to collaborate with researchers and developers worldwide. If you are an expert in technology, security, economics, governance, policy or design and are interested in helping build Skipjack into a global currency, and universities, open-source projects, companies and non-profits who are interested in partnering with Skipjack , and if you have feedback on Skipjack, please send ideas and suggestions to email at ceo@skipjackx.com