Formal Verification of Block Ciphers in HOL Theorem Prover

Ruofan Yang

October 24, 2024

Abstract

Interactive Theorem Proving (ITP) combines the strengths of manual and automated proofs. It is used to prove the correctness of intended algorithms in a more rigorous and formal manner, ensuring that there's no logical errors in the algorithm. The Data Encryption Standard (DES) is a block cipher which ever played a significant role in cyber security, and is one of the most widely used cryptosystem. DES has many well known properties such as the complementation, weak and semi-weak keys, which are common used in related work. This paper use the HOL4 interactive theorem prover to define the properties based on the built-in DES implementation and then prove these properties. The paper also includes an implementation of the RC5 cryptosystem. RC5 is a symmetric-key block cipher well known for its variable block and key size.

Furthermore, there is an initial implementation of the crypatanalytic attack method called *Differential Cryptanalysis* against DES, and we have proved some basic theorems about it. The work on Differential Cryptanalysis here is to break DES variants with fewer rounds. The work ensures the correctness and effectiveness of cryptosystem and attack method, enabling error-free, reliable operations in real-world applications.

1 Introduction

Cryptography has been widely used in the world since centuries ago. It is used for constructing protocols to protect the data and secure the communication. Block cipher is a deterministic algorithm. it uses fixed-length blocks that are composed with bits to encrypt and decrypt data. The Data Encryption Standard (DES) and RC5 are both famous block ciphers. They have been commonly used, studied, researched by many people, but people rarely question the correctness of some basic properties about them (instead, these properties are directly trusted and used). In rare cases (with a different newer cipher, e.g.), such unconditional trusts may cause significant vulnerabilities and insufficient security level of encrypting data, and thus the cryptosystem built upon them may fail to secure important data, where data breaches may occur.

In this paper, the author has verified some important properties of the block cipher DES and RC5, ensuring the correctness of these properties (thus provide direct evidence of the security and reliability of DES and RC5). This work mainly targets the correctness proofs of complementation property, weak key and semi-weak key property of DES, the implementation and encrypt-decrypt correctness of RC5, and an initial work as a set of theorems related to the verification of Differential Cryptanalysis. Differential Cryptanalysis is a type of cryptanalytic attack used in DES-like cryptosystems, it can break variant DES of up to 15 rounds. The initial built definitions and verifications of Differential Cryptanalysis ensures the correctness of basics, thus support the future construction. This paper use the Interactive Theorem Prover HOL4 to do all the implementation and verifications, it provides solid and large bases of theorems, built-in decision procedures and tactics,so I can build the verification based on them straightly and expediently. As a result, the paper also enboard the theory base of HOL4, provide some fundamentals of block cipher construction which allow future research and verification efforts in HOL4.

2 Preliminary

2.1 Types and terms in HOL4

HOL4 theorem proving system is a proof assistant with built-in decision procedures, tactics and theorems to be convenient to prove harder theorems for users. It is composed with HOL types, terms, rules of inference and theories. The types in HOL4 contains the basic atomic types like *int*, *bool* or the *wordn* type which is commonly used in proving cryptography aspect. It also contains the compound types such as *set* and *list* , and also the function types which are types of function from one type to another.

In HOL4. each term should have a type and a term can be a variable, constant and function. The rules of inference are used to derivate new, more complex theorems from existing ones. They are implemented as ML functions and take terms or theorems as input and return theorems as outputs, each can be treated as a proving step during the proving. Theories are composed with the type structure, signature, set of axioms and set of theorems. Each theory is an extension of the parent theories that focusing on some more specific aspects from its parent theories and research them in more details. It builds upon the concepts from the parent theories, and they are overall organized hierarchically to form a tree-like framework.

2.2 Basic use of HOL4

rw[] simp fs RW_TAC POP_ASSUME MP_TAC Q.ABBREV_TAC Abbr Suff Know Rewr'

2.3 Word theory in HOL4

In this paper, *wordsTheory* in HOL4 are mainly used to prove the properties of DES, Differential Crypatanalysis and the implementation of RC5. It is built upon some basic theories and *fcpTheory* which are also common used in this paper. The terms used in the verifications are mainly in type of wordn which means a word with length n. *wordsTheory* contains many definitions and theorems about the words' operations, analysis and properties.

There are some operations that are frequently applied. The *word_concat_def* defines the operator @@, it takes two inputs v and w of type word n and word m, join them into a word of length $(n+m)$, the first m bits are the same as w, and the left n bits are the same as v.

$\vdash$ v `@@` w `= w2w (word_join` v w`)`

Then the $word_extract_def$ defines the >< operator, it uses two number inputs h and l where $(h > l)$, and is applied to a word. It extracts from the lth bits to the hth bits of the word, result a word of length $(h - l + 1)$

$\vdash$ `(`h `><` l`) = w2w` $\circ$ `(`h `--` l`)`

The ?? operator defined by *word_xor_def* also takes two words of same length as inputs, each corresponding bit of the two words are applied with the xor (Exclusive-OR) operation and result a words of that length.

$\vdash$ $v \oplus w$ `= FCP` i. v ' i $\not\Leftrightarrow$ w ' i

At last, one common operator used in proving the property of complementation is , it takes a word as input, and converts each bit to the inverse (1 to 0 and 0 to 1 for binary form), produce the bitwise complement of word as result.

$\vdash$ $\neg w$ `= FCP` i. $\neg w$ ' i

The *wordsTheory* is built with the support of *fcpTheory*, *fcpTheory* build the foundation that enables the bit-level operations. Consequently, we can do the operations mentioned above as they need to analyze bits of a word.

2.4 Measure and Probability theory in HOL4

The *measureTheory* and *probabilityTheory* build the definitions of algebra $(X\ ,\ A)$, $\sigma - algebra$, thus the measure space (X,A,u) and probability space. The $\sigma - algebra$ is algebra that the union of all subsets of A also belongs to A. Probability space is a measure space satisfy the property that $u(X)\ =\ 1$, and are often represented as $(\Omega\ ,\ F\ ,\ P)$. The theories also includes the complementary definitions and theorems related to the properties of these definitions. The theories are built upon the *extrealTheory*, *extrealTheory* extends the range of standard real number, adds the positive infinite and negative infinity. It is because algebra is a set which include not only finite algebra but also infinite algebra.

```
[measure_space_def]
 ⊢ measure_space m ⟺
   sigma_algebra (measurable_space m) ∧ positive m ∧
   countably_additive m
```

The *measureTheory* and *probabilityTheory* are used in the implementation of Differential Crypatanalysis as the need of working with sets. It requires to get the probabilities of subsets out of a universal set. In the cases of Differential Crypatanalysis attack against DES, the overall sets are usually the universal sets of words or word pairs of a specific length, the target sets are DES inputs of that length and satisfy particular properties.

3 Data Encryption Standard (DES)

Data Encryption Standard (DES) is a block cipher that inputs 64-bit blocks with a 56-bit key[3]. It is a Feistel cipher, consisting of 16 rounds of encryption that repeat the same encryption procedure for each round. The key entered by the users are 64-bit, but 8 parity bits are ignored. The round keys used in each round are implemented by the Permuted Choice 1 (PC1) first, it splits the 64-bit key to two 28-bit words with rearranging and ignores the 8 bits in positions $8 * n\ th$ of the key, load them to two registers C, D. Then to compute round keys in different rounds, C and D are rotated by one or two bit positions to the left in each round, therefore the two 28-bit words are different in different rounds. Finally, round keys are 48-bit keys that are extracted from Permuted Choice2 (PC2).

Next, to encrypt a plaintext, the input message first undergoes an initial permutation (IP) and then splits to two 32-bit halves. Then the halves u,v are passed to the round process, one of the halves v is applied to the round function with round keys, undergoes XOR operation with the other half u. The result 32-bit word v1 is swapped with v, so v1 will be the input of the round function in the next round and XOR with v, the process will repeat for 15 times. However, in the last round, two output halves will not be swapped, they are joined, undergoes the inverse permutation IP^{-1} to produce the final 64-bit output.

The round functions in each round is the same, they first expand the 32-bit half to 48 bits, then operate it with round key by XOR operation, the 48-bit intermediate value are separated into eight 6-bit words, put each into a different S-boxes, and the eight outputs of 4-bit are concatenated to form a 32-bit message. At last, the 32-bit message is permuted by A bit-level permutation (P) and produces the output of the round function.

The S-boxes are the only nonlinear components in DES [3]. Each S-box can be considered as a table of 4 rows and 16 columns, containing values from 0 to 15. the 6-bit input is split, two outer bits are formed to choose the row of table and the four inner bits are formed to get the column, the value in that row and column is the 4-bit output of that S-box.

It is worth noticing that DES, as a Feistel cipher, has a decryption process that follows the same procedures as encryption but use the round keys in reverse order.

4 DES properties

In this paper, some important DES properties are implemented and proved with the assistance of HOL4. They are the implementation properties, weak key and semi-weak key properties.

The DES complementation property [2] is for a plaintext input m and key k, and the bitwise complement of them m and k, the complement result after encrypting m using DES and k as key

$$\overline{\mathrm{DES}_k(m)} = \mathrm{DES}_{\overline{k}}(\overline{m}).$$

Figure 1: Formula for DES complementation property

$$\mathrm{DES}_w(\mathrm{DES}_w(m)) = m.$$

Figure 2: Formula for Weak key property

is the same as the result after the DES encryption step using w and k like shown in Figure 1. By verifying this, we know a limitation of DES, in a chosen message attack this property can reduce the cost of exhaustive key search by half[2]. It is because to check if one guessing k is right, one known message and DES encryption can produce two ciphertext c and c by applying bitwise complementation to c. This process equals to two DES encryption for two known messages and keys, the exhaustive search of key of length 56 can reduce from 2^{55} to 2^{54}.

Weak keys are four special keys follow the property that DES encrypts a message twice with the same weak keys will return the same message shown in Figure 2. It is because the weak keys cause round keys equal to the reverse of the round keys, cause the DES encryption and decryption to be the same. Weak keys also have the property that each weak key has corresponding 2^{32} messages that encrypt these messages with that weak key will result message unchanged. These messages have the property that their input halves are the same in the encryption process of the 8th rounds. By combining this with the weak keys' reverse order round keys property, it ensures the input halves in the (8-n)th and (8+n)th round are the same, result the original message (0th round) the same as the ciphertext (16th round).

Similarly, semi-weak keys are six key pairs that encrypts a message with one of the key in the pair then encrypts the resulting ciphertext with the other key in the pair will keep the message unchanged. The formula is shown in Figure 3. The reason is the same as weak keys, the round keys of the two semi-weak keys in a pair are arranged in the reverse order.

These properties cannot reduce the complexity for exhaustive key search especially for only 4 weak keys compare to the total of 2^{56} keys. However, their existence can be vital when DES is used within some construction[3].

4.1 DES in HOL4

DES algorithm has been built in the *desScript* file in HOL4, and my paper build the verifications based on the DES implemented in this file.

In the file, it first defines some specific types used in the Data Encryption Standard implementation. It includes the halves of word/plaintext as a pair of word32, halves of keys in each round as pair of word28 and a function type of S-Box that take word6 as inputs and return word4. It also includes the data tables to help facilitate the IP, inverse IP permutations, E expansion, P, PC1 and PC2 permutations, round-dependent rotation values and the permutation values for the S-boxes. Then there are some functions and definitions that perform the expansion, permutations, reversion and the S-Boxes' processing. The "Key Scheduling" section in this file are definitions to build the round keys from the input key. As the last part of preparing, there are join and split functions to initially split the plaintext and form the result ciphertext. Theorems are also constructed to verify some basic properties and help ensure the correctness of implementations.

It then actually build the XOR operation, round function and swap operation in each round. Moreover, it combines the above operations, along with the round keys to form the actual process for each round and ultimately construct the complete DES process over a total of16 rounds. It also

$$\mathrm{DES}_{w_1}(\mathrm{DES}_{w_2}(m)) = m.$$

Figure 3: Formula for Semi-weak key property

implements a help function that can return the input halves in each round as a pair in the form of (M a, M b). It is beneficial for any future analysis of DES, including the verification of the proper DES implementation in *desScript* and my additional DES properties proving. At last, it proved that as a Feistel network, the implemented encryption of DES using the round keys, followed by a second encryption using the reversed round keys will return the original plaintext.

Consequently, it not only verifies that the decryption of DES is the same process of encryption but with reversed round keys, but also confirms the correctness of DES implementation in HOL4 by proving that decrypting the encrypted message with the same key will return the original message.

5 DES property in HOL4

In my *des$_p$ropScript* file in HOL4, there are some basic settings. It loads all the relevant theories and libraries which contains necessary definitions and theorems I need to use including the *desScript*. I set the *guessing_word_lengths* to true so that I do not need to set the length for each word, they will be assigned to a length based on the conditions. Additionally, I also define a simpleset *fcp_ss*, it is a simplification set that combines *std_ss*, the basic simplification set, with *FCP_ss*, the simpset fragment specifically simplifies finite Cartesian product expressions.

5.1 Complementation property

First of all, I proved the complementation property, it is implemented as the theorem *comple_property*.

$$\begin{aligned}&\vdash 0 < n \land n < 17 \land \texttt{DES}\ n\ k = (\mathit{encrypt},\mathit{decrypt}) \land\\ &\quad \texttt{DES}\ n\ (\neg k) = (\mathit{encrypt}',\mathit{decrypt}') \Rightarrow\\ &\quad \neg\mathit{encrypt}\ m = \mathit{encrypt}'\ (\neg m)\end{aligned}$$

It means for any key k, message m and number of rounds n, the complementation of encryption using m and k is the same as the encryption using the complementation of m and k.

I initially convert the *encrypt* function to more detailed components functions including inverse of initial permutation, join, swap, round, split and initial permutation functions. This allows the use of rewriting rules on separated functions. This includes some theorems proving the complementation property of each individual function mentioned above. The below theorem is an example for join function.

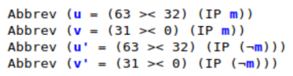
$$\vdash \texttt{Join}\ (\neg m,\neg n) = \neg\texttt{Join}\ (m,n)$$

I then convert the *Round* function to the $(M\,a,\ M\,b)$ by the *half_message* help function, so that I can research on each 32-bit half of the block and use the relation between halves in the same and different rounds. Moreover, I simplify the proving goals by rewriting it with the theorems about the complementation property of each individual function, leave only solving the complementation property of a pair of M function.

$$\begin{aligned}&(M\ (u',v')\ \mathit{keys}'\ n,\ M\ (u',v')\ \mathit{keys}'\ (\mathit{SUC}\,n))\\ &\qquad = (\sim M\ (u,v)\ \mathit{keys}\,n,\ \sim M\ (u,v)\ \mathit{keys}\ (\mathit{SUC}\,n))\end{aligned} \tag{1}$$

I decide to use the Mathematical induction method using *Induct_on* in HOL4. I replace the variable n to x, define x to be <= n, and then perform induction on variable x. It is to avoid the failure of proving the induction step. As the prerequisite of using given case is $n > 0$, but in the step case, only $(SUCn) > 0$ (SUC means successor) is given. From $n + 1 > 0$, it is not possible to derive the requirement of $n > 0$, thus given case cannot be used, verification cannot be pushed further. By converting to x, the prerequisites can be easily meet, and facilitating the proof process.

To prove the base case, we only need to convert back to the form of *Round* function and use the base case of the *Round* function. It is because only *Round* function is built using the relationships between its values based on inductively given inputs, and it has the base case for $x = 0$. To complete the base case verification, I also prove $(u,\ v) = (u',\ v')$ by extract the abbreviation of them back into their original forms, and apply the complementation rule of IP function. The abbreviation can be viewed below 4.



```
Abbrev (u = (63 >< 32) (IP m))
Abbrev (v = (31 >< 0) (IP m))
Abbrev (u' = (63 >< 32) (IP (¬m)))
Abbrev (v' = (31 >< 0) (IP (¬m)))
```

Figure 4: The abbreviation

It is important to note that $(M\,a,\ M\,b)$ can be converted from the *Round* function. It is defined as a pair of two halves taking nth round as one of the inputs, which are the two halves in the n-th round during the DES encryption process. According to the workflow of encryption, the right half of round n equals to the left half of round $n+1$ by the swap operation at the end of each round. Depend on this and the workflow, the right half of round $n+1$ can be expressed as the two halves in previous round with some operations. By using the relationship of halves between different rounds, each half of $(M\,a,\ M\,b)$ in round $n+1$ can be expressed by the halves in round n. Consequently, in the induction step, the step case which contains $(M\,a,\ M\,b)$ in round $SUC\,n$ can be expressed into formulas using $(M\,a,\ M\,b)$ in round n which are contained in the base case. As a result, I can use the assumptions provided in the base case to push forward my verification.

$$\begin{aligned} &Assumption: \quad (M\ (u',v')\ keys'\ x,\ M\ (u',v')\ keys'\ (SUC\,x)) \\ &\qquad = (\sim M\ (u,v)\ keys\ x,\ \sim M\ (u,v)\ keys\ (SUC\,x)) \end{aligned} \tag{2}$$

$$\begin{aligned} &Goal: \quad (M\ (u',v')\ keys'\ (SUC\,x),\ M\ (u',v')\ keys'\ (SUC\,(SUC\,x))) \\ &\qquad = (\sim M\ (u,v)\ keys\ (SUC\,x),\ \sim M\ (u,v)\ keys\ (SUC\,(SUC\,x))) \end{aligned} \tag{3}$$

By the above conversions and simplifications using base case, the proving goal is now transformed to verify only the complementation property about round function. It is to prove that round function with 32-bite message m and round key generated by key k equals to the round function with inputs of the complementation of both m and k. In HOL4, *RoundOp* function is built to represent round function of each round combining the expansion E, XOR operation with round key, S-boxes permutations S and the bit-level permutation P. By expanding the *RoundOp* function, the goal can now reduce to:

$$\begin{aligned} &P\ (S\ (E\ (\sim M\ (u,v)\ keys\ (x+1))\ \oplus\ (EL\ x\ keys'))) \\ &\qquad = P\ (S\ (E\ (M\ (u,v)\ keys\ (x+1))\ \oplus\ (EL\ x\ keys))) \end{aligned} \tag{4}$$

As the property of exclusive OR operation, $\sim\ a \oplus\ \sim\ b\ =\ a \oplus b$ and the support complementation property of the expansion function, $E\ (\sim\ a)\ = \sim\ E\ (a)$ once I prove the complementation of round key generated by key k is equal to the round key generated by the complementation of key k, the verification of complementation property in DES is done.

The implementation of the design about round key in HOL4 is generated by *RoundKey* and *KS* function. *RoundKey* outputs a round key list that the elements in it are pairs of 28-bit words, each pair represents the two halves after the splitting of initial key, the rearrangement by PC1 and the rotation in different rounds. *KS* then using the *MAP* function to map each element in the output of *RoundKey* to perform PC2 permutations. For convenience in the verification, I define three functions *RK*, *RK_L* and *RK_R*. *RoundKey* can be converted to *RK* and thus split into $(RK\_L\ a\ , RK\_R\ a)$, and now I can work on each half of the pair separately, instead of only be able to work on the pair as a whole. The current form of goal can be expressed like:

$$\begin{aligned} &EL\ x\ (MAP\ f\ (list\ of\ input\ \sim k)) \\ &\qquad = \sim EL\ x\ (MAP\ f\ (list\ of\ input\ k)) \end{aligned} \tag{5}$$

By applying a series of rewriting rules in *listTheory* that can deal with EL, MAP and some functions in f, the complementation theorems for *RK_L* and *RK_R*, the goal finally transforms to $MAP\ f\ l = MAP\ f'\ l$. At last, the complementation property of PC2 permutation is used to finish the proof of $f = f'$. The initial goal of DES complementation property is proved in HOL4 through the above steps.

As mentioned above, I proved many support theorems about the complementation property of each single operation, their proofs are done in a similar way. The operations are expanded by their definitions to transform the verification into each bit of $\sim m$ at the permuted indices equals to the complementation of each bit of m in the same indices. For example, the indices of IP permutation are transformed to $64 - EL\ (63 - i)\ IP_{data}$ where i is the original index of m before IP permutation. IP_{data} is the IP table to do the permutations, and it rearranges the bits in different positions of m to form a new message. Then I need to prove that the transformed indices are still less than the length of input message m to meet the prerequisite for using the theorem, FCP_BETA and then I can complete the proof.

$\vdash i <$ `dimindex` $(:\beta) \Rightarrow$ `(FCP)` g `'` i `=` $g\ i$

The proof that the transformed indices are all less than the length of input message m (64) given i is less than 64 uses the method of exhaustion. It repeatedly proves the transformed index is less than 64 for each value of i for 64 times, covering all cases from i=0 to i=63. The proof is completed as all 64 situations meet the condition. Overall, the support theorems are all proved with a similar way.

There are some special built-in tactics, conversions used here. The $MATCH\_MP\_TAC$ tactic takes a theorem as input, it can be applied to the goal if the current goal is in the form of the consequent of this theorem. The goal can then be converted to the antecedent of the theorem. rpt is a tactic that takes a tactic as input and repeatedly apply the tactic until it no longer succeeds. $CONV\_TAC$ is a special tactic that makes a tactic from a conversion. Then $BOUNDED\_FORALL\_CONV$ is a conversion that deal with universal quantification for bounded natural numbers, so it converts $\forall n.\ n < k$ to the conjunction of $n = 0$ to $n = k-1$, like the exhaustion methods for i that I mentioned above.

There are also some theorems and definitions that are frequently used in the proof. $word_1comp_def$ defines the complementation of a word m $\sim m$ equals to the concentration of the complementation of each bit of word m. It helps deal with the complementation in bit level.

$\vdash \neg w$ `= FCP` $i.\ \neg w$ `'` i

When dealing with the lists and some operations applies to lists like the map function. MAP_TL shows that applying a function to the tail of a list is equal to the tail of a list after the mapping. Then the theorem MAP_MAP_o states applying two functions to a list using MAP twice is equivalent to applying the composition of the two function using MAP once.

$\vdash$ `MAP` f `(TL` l`) = TL (MAP` $f\ l$`)`

$\vdash$ `MAP` f `(MAP` $g\ l$`) = MAP (`$f \circ g$`)` l

5.2 Weak and semi-weak key property

I first define the four weak keys into HOL4 in hexadecimal form. Then I want to prove in HOL4 that for all the DES encryptions using the weak keys, applying the encryption twice to any message will result the message nochange.

$\vdash$ `MEM` k `Wkey` $\land$ `FullDES` k `= (`$encrypt$`,`$decrypt$`)` $\Rightarrow$
$encrypt$ `(`$encrypt\ plaintext$`) =` $plaintext$

It can be proven easily using the existing theorem $desCore\_CORRECT$ in $desTheory$. DES has the property that the decryption in DES of a key is the same process of encryption in DES but use the reverse list of round keys generated from the same key. The $desCore\_CORRECT$ theorem uses this property and the fact that decrypting an encrypted message returns the original message. It successfully proves that encryption to a message twice with a key, once using the normal round keys, once using the round keys in reverse order remains message unchanged.

$\vdash$ `LENGTH` $keys$ `=` $r \Rightarrow$
`desCore` r `(REVERSE` $keys$`) (desCore` $r\ keys\ plaintext$`) =`
$plaintext$

By applying this to the goal, I am left with only the proofing of the prerequisite of *desCore_CORRECT* and the proof of round keys list generated by weak key remains the same after being reversed, that is:

$$REVERSE\ (KS\ k\ 16) = KS\ k\ 16 \tag{6}$$

The rest part is easy to prove, the length of the list of round keys is equal to the number of rounds by the *LENGTH_KS* theorem. The equivalent proof about the round keys can be done by substituting each weak key's value into the goal, use the *EVAL_TAC* that applies the values of keys to the goal and evaluate on the value. The goal is proved after the successful equation calculation on each value.

$\vdash$ `LENGTH (KS` k n`) =` n

The semi-weak keys' property and its corresponding verification are similar to weak keys'. I define them as pairs in a list, set the goal to be: encrypting a message with one key in a pair and encrypting the ciphertext with another key in that pair make the ciphertext back to the original message.

$\vdash$ `MEM` *pair* `Semiwkey` $\land$ *pair* `= (`s_1`,`s_2`)` $\land$
`FullDES` s_1 `= (`*encrypt*`,`*decrypt*`)` $\land$
`FullDES` s_2 `= (`*encrypt*$'$`,`*decrypt*$'$`)` $\Rightarrow$
encrypt `(`*encrypt*$'$ *plaintext*`) =` *plaintext*

$\vdash$ `MEM` *pair* `Semiwkey` $\land$ *pair* `= (`s_1`,`s_2`)` $\land$
`FullDES` s_1 `= (`*encrypt*`,`*decrypt*`)` $\land$
`FullDES` s_2 `= (`*encrypt*$'$`,`*decrypt*$'$`)` $\Rightarrow$
encrypt$'$ `(`*encrypt* *plaintext*`) =` *plaintext*

The way to achieve this goal only change the part about the equivalent proof of round keys and reversed round keys. It is now to verify that the round keys generated by one of the semi-weak keys are equal to the revered round keys generated by the other key. The method of verification is still use *EVAL_TAC* to evaluate on the values of semi-weak keys. The proof is then done.

$$REVERSE\ (KS\ k1\ 16) = KS\ k2\ 16 \tag{7}$$

5.3 Fixed points for DES weak keys

I can now prove each weak key has 2^{32} messages that encrypting these messages with the corresponding weak key remain the messages unchanged. As mentioned in earlier section, these messages have this property based on their property that the input halves are the same in round 8 during the DES encryption using the corresponding weak key. Based on this, I define four sets for four weak keys that each list obtains messages with the above property. I can then prove separately that all messages in the set meet the property that encryption does not change the message, and each set has cardinality of 2^{32}.

$\vdash$ x $\in$ `Wtext1` $\land$ `FullDES Wkey1 = (`*encrypt*`,`*decrypt*`)` $\Rightarrow$
encrypt x `=` x

$\vdash$ x $\in$ `Wtext2` $\land$ `FullDES Wkey2 = (`*encrypt*`,`*decrypt*`)` $\Rightarrow$
encrypt x `=` x

$\vdash$ x $\in$ `Wtext3` $\land$ `FullDES Wkey3 = (`*encrypt*`,`*decrypt*`)` $\Rightarrow$
encrypt x `=` x

$\vdash$ x $\in$ `Wtext4` $\land$ `FullDES Wkey4 = (`*encrypt*`,`*decrypt*`)` $\Rightarrow$
encrypt x `=` x

First, to prove DES encryption with messages in the set remains message unchanged, I expand the goal to more detailed operations during the DES encryption process. I find that only if I proved that the reverse of the halves in round 16 during the encryption process are equal to the halves in round 0, the rest of the proof is quite simple. I can use the halves in round 0 which do not process any round function, but only the inital IP permutation and splitting to simplify the goal to:

$$IIP\ (Join\ (Split\ (IP\ x)))\ =\ x \tag{8}$$

These operations can be easily cancelled out by each other using theorem *IIP_IP_Inverse* nad *Join_Split_Inverse*.

```
⊢ IIP (IP w) = w
```

```
⊢ Join (Split w) = w
```

I am now left to prove the support theorem *weakK_sup* given the property of each set. It is hard to compare two pairs of halves that differ by sixteen rounds of operations directly. To achieve the goal in HOL4, I build the support theorem *weakK_sup* by setting the goal to show that for be each pair of halves in (8-n) round is equal to the reversed pair in round (8+n). I can then use proof by induction on n to complete the proof.

```
 ⊢ MEM k Wkey ∧ 0 ≤ n ∧ n ≤ 8 ∧
Split (IP (desCore 8 (KS k 8) x)) = (w,w) ⇒
Round (8 − n) (KS k 16) (Split (IP x)) =
Swap (Round (8 + n) (KS k 16) (Split (IP x)))
```

As I define the message sets using round keys list contains only up to the $8th$ round to facilitate future verification, I first prove that doing the encryption with only 8 round keys is the same as encrypt with all 16 round keys when just encrypting for 8 rounds by another support theorem. By using it, I convert the prerequisite to mathch the round keys list with the goal.

Then I start the induction on n and the base case is to prove the two equal input halves (w , w) at round n is the same after swapping. It is straightforward, and I only need to expand the *Round* function to pair form and apply the *Swap* definition.

For the induction step, I know the base case shows that the reversed halves in round a (Ra , La) are equal to the halves in round b (Lb , Rb) where a and b represents $8+n$ and $8-n$. To proving the step case, I can directly get the right half of round $(b-1)$ $R(b-1)$ is equal to the left half of round b Lb and the left half of round $(a+1)$ $L(a+1)$ is equal to the right half of round a Ra because only swap operation happens on these halves between rounds. As $Ra\ =\ Lb$ by the base case, I can get $R(b-1)$ and $L(a+1)$ are the same. Then because the encryption and decryption process of DES are the same, I can get $L(b-1)$ back from halves in round b through the round function $f(b-1)$ by using the swap operation, round function and XOR operation

$$L(b-1)\ =\ (Rb\ \oplus\ f(b-1)\,(\ Lb\ ) \tag{9}$$

$R(a+1)$ can also be represented by Ra and La in normal encryption manner through the round function $f(a+1)$

$$R(a+1)\ =\ La\ \oplus\ f(a+1)\,(\ Ra\ ) \tag{10}$$

According to the proof done above that round keys list generated by a weak key remains the same after being reversed, I can conclude that the list is symmetric, and thus the $(b-1)$th and $(a+1)$th round keys are the same in the round keys list of length 16. The round function $f(b-1)$ and $f(a+1)$ are then the same as round key is the only change variable for round functions in different rounds. By combining this with $Rb\ =\ La$ and $Lb\ =\ Ra$, $L(b-1)$ and $R(a+1)$ are the same, and we can prove the step case is also right given the base case.

Using the above way in HOL4, I convert the goal to halves using ($M\ a$, $M\ b$) function and then express the step case' halves using the base cases' halves. The only goal left is the proof of the elements in symmetric positions are the same. I need to put each weak key's value into the goal, calculate the generated round keys list for each key, ensure the step case's index do not exceed the length of list. At last, I need to do the method of exhaustion, examine each possible value of n from 0 to 15 to check if they meet the goal because of the lack of theorems in HOL4 on this aspect. The support theorem is proved.

$$EL\ (7-n)\ (KS\ k\ 16) = EL\ (n+8)\ (KS\ k\ 16) \tag{11}$$

Switch to the last support theorem *wkey1_equal* mentioned above. It is also proved by induction, the base case can be directly proved by the base case of *Round*. The indcution step is proved in the same way as above by transforming the *Round* function to ($M\ a$, $M\ b$) form first and then express the step case' halves using the base cases' halves. The last part of verification is also examine each possible value of n to the goal, use the method of exhaustion to test all situations meet the goal and the theorem is proved.

```
⊢ MEM k Wkey ∧ n ≤ 8 ⇒
Round n (KS k 8) (Split x) = Round n (KS k 16) (Split x)
```

It is time to prove that the cardinality of the sets of these messages is 2^{32}. I show these sets form bijection with the universal set of 32-bit words first. To achieve these, I define two functions for each set, one of the two can map the messages set to the universal set and the other can map reversely.

```
⊢ BIJ w1trans1 Wtext1 𝒰(:word32)
```

```
⊢ BIJ w2trans1 Wtext2 𝒰(:word32)
```

```
⊢ BIJ w3trans1 Wtext3 𝒰(:word32)
```

```
⊢ BIJ w4trans1 Wtext4 𝒰(:word32)
```

To prove the bijection relation between the two sets, *BIJ_IFF_INV* theorem can be used to convert the *BIJ* function into four sub-goals. It includes proving that all elements in messages set belongs to the other universal set of word32 after mapping, and vice versa. Moreover, the other two goals are applying the two functions together to the sets will keep the elements unchanged.

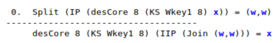
```
⊢ BIJ f s t ⟺
(∀x. x ∈ s ⇒ f x ∈ t) ∧
∃g. (∀x. x ∈ t ⇒ g x ∈ s) ∧ (∀x. x ∈ s ⇒ g (f x) = x) ∧
    ∀x. x ∈ t ⇒ f (g x) = x
```

Taking the example for proving the first weak key, and the other weak keys are all proved in the same way. Their corresponding sets and functions just change the weak key used. The first sub-goal that all elements in messages set belongs to the other universal set of word32 after function $w1trans1$ is proved directly. It takes the 64-bit messages as inputs and outputs only the first reversed halves of message encrypted by 8 rounds. As the outputs are 32 bits, they surely belong to the universal set of word32.

```
⊢ w1trans1 x = FST (Split (IP (desCore 8 (KS Wkey1 8) x)))
```

The second sub-goal can be converted using definition of function $w1trans2$ and set $Wtext1$. All 32-bit messages (in universal set of word32) are formed as a pair of equal halves, applied with *Join* and IIP operation, encrypted for 8 rounds twice, and applied with IP and *Split* operation. And to prove the result halves are still the same.

```
⊢ w1trans2 x = desCore 8 (KS Wkey1 8) (IIP (Join (x,x)))
```



```
 0.  Split (IP (desCore 8 (KS Wkey1 8) x)) = (w,w)
------------------------------------
     desCore 8 (KS Wkey1 8) (IIP (Join (w,w))) = x
```

Figure 5: Apply two functions to elements in Wtext1

$\vdash$ `Wtext1 =`
$\{x$ `|` $\exists w$. `Split (IP (desCore 8 (KS Wkey1 8)` x`)) = (`w`,`w`)`$\}$

I can also use the previous method to help the proof, the two encryptions of 8 rounds with the weak key cancel each other out as one of the encryption can be treated as decryption. The *Split* and *Join* operations can be cancelled out, and also the *IP* and *IIP* operations, results the initial pair of haves unchanged, so the two halves must remain the same with no change. The second sub-goal is then proved.

The third sub-goal is to prove for an element in the messages' set with the described property, when they are converted tp the 32-bit half through $w1trans1$, formed as a pair of equal halves and map with the $w1trans2$, the result messages remain the same 5. It proves with the same way as sub-goal 2, but begins to cancell out the *Join* and *Split* operation first, then the *IIP* and *IP* operations and at last the two encryption processes with weak key. It also cancells each other out, but in a different order.

The elements are initially in the universal set, change to pairs equal halves, and apply the $w1trans2$ function followed by the $w1trans1$ function in the last sub-goal. It can also be verified by cancelling each operations in another order. The bijection relation between two sets is finally proved.

The bijection proofs for the other three keys are exactly the same, so one proof *BIJ_for_weak_keys_explicit* with the lists of sets, functions and weak keys as inputs can be done using the existing seperate BIJ theorems.

$\vdash i < 4 \Rightarrow$ `BIJ (EL` i `wtrans1) (EL` i `Wtextlist)` $\mathcal{U}$`(:word32)`

The universal set of word32 has the cardinality of 2^{32} by the theorem
card_word32. As a result, by proving that these sets have the same cardinality as the universal set of word32, I can then prove the cardinality of each message set is 2^{32}. I can meet the prerequisites that the universal set of word32 is finite directly and use the above bijection relation, then theorem *FINITE_BIJ_CARD* can be used to prove the equivalence of cardinality.

$\vdash$ `CARD` $\mathcal{U}$`(:word32) = 2 ** 32`

$\vdash$ `FINITE` $s \land$ `BIJ` $f\ s\ t \Rightarrow$ `CARD` s `= CARD` t

As a result, each message set is proved with cardinality of 2^{32} and the theorem is stored as *DES_weak_fp_card* in *des_propTheory*.

$\vdash$ `MEM` x `Wtextlist` $\Rightarrow$ `CARD` x `= 2 ** 32`

There are also some new tactics used here to help complete the goal.
Q.PAT_X_ASSUM (*Assump*) *MP_TAC*, it takes a assumption of current goal *Assump* as input, find the first matching assumption, remove the assumption and take it as a prerequisite for the current goal. It has similar use as *POP_ASSUM MP_TAC* but avoid the repeat usage of
POP_ASSUM MP_TAC when there are too many assumptions.

The *Q.EXISTS_TAC* tactic can be applied to existentially quantified goal. It can substitute a specific term I decide u to the existentially quantified variable $?x$ in the goal. The proof is now changed on u.

6 Differential Cryptanalysis in HOL4

Differential crypatanalysis of DES-like cryptosystem is a type of cryptanalytic attack which can break reduced variant of DES up to 15 rounds.[4] Differential crypatanalysis is a chosen-plaintext attack, it can only be used when attackers are able to encrypt any plaintext using DES encryption and get the corresponding ciphertexts. It can break any variant DES by attempting the DES encryptions for at

Figure 6: Differential cryptanalysis for S-boxes

most 2^{56} time with at most 2^{56} plaintexts. The Differential cryptanalysis attack is developed by the differences in plaintext pairs and differences of ciphertext pairs.

As my current research on differential cryptanalysis is just beginning, the following discussion will be limited within a single round of DES. The plaintext pairs are two 32-bit message inputs that are passed through the round function in a round. The ciphertext pairs are the outputs of two round functions using the plaintexts pair in that round. As S-boxes part is the only nonlinear components of the cipher in DES [3], for pairs of plaintext with fixed exclusive-or differences, the differences will only change to uncertain values through S-boxes. Other than S-boxes, the expansion operation and the bit-level permutation P change the XOR value to another fixed *values*. For the XOR value of two messages after applying XOR operation to them separately with the same round keys, the result value remains the same difference value. These properties are implemented and proved in HOL4.

$\vdash$ `P` $x_1 \oplus$ `P` x_2 `=` `P` $(x_1 \oplus x_2)$

$\vdash$ `E` $x_1 \oplus$ `E` x_2 `=` `E` $(x_1 \oplus x_2)$

The way to prove them is similar to the proof of the complementation property of these operations. I want to prove for each bit of x1 and x2 after the permutation, each exclusive-or value of the bit pairs is equivalent to the bit value after the permutation for exclusive-or x1 and x2. The first thing needs to do is to check if the permuted bit indices are still less than the length of words. Then I expand the definitions of P and E operations, bring in the P and E permutation tables, and use the method of exhaustion to test on each bit from 0 to the length of output. Since the permuted indices are all less than the length in all cases, the verification is succeeded. At last, by using the definition of XOR in HOL4 that state applying XOR operation to two words means applying XOR operation to each bit of two words and FCP_BETA that simplify the FCP out, the proof is done.

$\vdash v \oplus w$ `=` `FCP` $i.\ v$ `'` $i \not\Leftrightarrow w$ `'` i

Proving *xor_twice* theorem uses the commutative property of XOR and the property that applying XOR operations to a value with itself will cancel out the value.

$\vdash (a \oplus b) \oplus c = a \oplus b \oplus c$

$\vdash$ `UINT_MAXw` $\oplus\ a = \neg a \wedge a \oplus$ `UINT_MAXw` $= \neg a \wedge$ `0w` $\oplus\ a = a \wedge$
$a \oplus$ `0w` $= a \wedge a \oplus a =$ `0w`

To prove the S-boxes are not linear is much simpler. I choose two words $0b0w$ and $0b0w$ to prove that there exist values showing S-boxes do not have additivity for XOR operation.

As the non-linearity of S-boxes, for pairs of plaintexts with the same XOR difference, the output XOR differences are different after the S-boxes if the pairs of plaintexts are different. I can then find a set of pairs of $XORed$ texts $(z\ ,\ z')$ that the difference value between them and the difference value after the S-boxes are fixed α and β. As the additivity for XOR operation in expansion function, I can directly use the XOR difference value after expansion. It will not affect anything important, but only change the input difference value that I choose to another fixed value by expansion and change the corresponding set. Similar to the ouput difference values I choose, they are values before the permutation P. As the additivity for XOR operation in it, only the inputs used to generate the sets are changed and the effect can be eliminated by changing the inputs. The inputs are chosen like that for facilitating the process and match with the table provided[4]. 6

Using only two plaintexts with fixed difference value and two encryptions, I can get a fixed output difference value. With the above two difference values, I can get a set of possible pairs of $XORed$ texts, and then I can find all possible round keys according to the set of possible pairs of $XORed$ texts and the two input plaintexts. Because apply XOR operations to two plaintexts x and $x1$ with two keys k and $k2$ will result the $XORed$ texts, I can then get the possible keys using the XOR operations between the plaintexts and $XORed$ texts by xor_trans. By keeping trying new plaintexts and then generating corresponding $XORed$ texts set, I can keep narrowing down the range of possible keys.

Differential Cryptanalysis implementation However, each S-box has 2^6 possible input $XORs$, 2^4 possible output $XORs$ and there are 2^6 pairs of plaintexts with each possible input XOR. For a total of 8 S-boxes, the tables that records the possible $XORed$ texts are too complex and the implementation of the real attack in HOL4 is not necessary for now. As a result, I implement the sets of $XORed$ plaintexts for S-box one and for the overall round function in a round. I prove the basic correctness of the S-box one set, and verify properties related to them.

First of all, I proved that a set of pairs of words of length 6 with fixed XOR value is bijective to the set of words with length 6. They are also proved to have the same cardinality, it can verify the property of pairs XOR distribution table of the S-box. It states that each line in a pairs XOR distribution table contains 64 possible pairs, means each particular input XOR has 64 possible pairs of word 6 inputs. I define sets of pairs of word 6 that one half has the fixed difference value of X with the other half. As the set of the universal set of word 6 has the cardinality of $2^6 = 64$, I can then prove the pairs sets also has the cardinality of $2^6 = 64$.

⊢ `BIJ trans2` $\mathcal{U}$`(:word6) (AllpairXor 0w)`

⊢ `CARD (AllpairXor 0w) = 2 ** 6`

By defining two mapping functions, one just returns a half in pair given the pair, to map the universal set of word 6 to the pairs set. The other function take the X difference value as input, and return a pair that one half is any word6 message, the other half is the word6 message that differs from the first by X. It maps the functions in another order. By using the BIJ_IFF_INV theorem, I can convert the goal to four separate sub-goals about mapping elements between functions. All sub-goals can be proved directly by applying the definitions of sets and mapping functions to the goal.

⊢ `trans1` (x_1,x_2) `=` x_1

⊢ `trans2` x `=` (x,x)

Next, I implement the definition of "X may cause Y with probability p by an S-box"[4] in HOL4. It means for all pairs of $XORed$ texts that the input XOR is X and the output XOR after going through a S-box is Y, the number of these pairs is p percentage out of all possible pairs of $XORed$ texts that the input XOR is X. These $XORed$ texts are in word 6 as the inputs of S-box and the input XOR is X means the pairs of texts is differ by X. The overall pairs of $XORed$ texts that the input XOR is X can then be form as the set of pairs of words of length 6 with fixed XOR value and the number of them is the cardinality of this set. As proved above, the set can then be converted to universal set of word 6 with the same functionality.

Then for the implementation of sets of possible pairs of $XORed$ texts with both input XOR X and output XOR Y, I find it can also convert to sets of single 6-bit texts that takes the X, Y and the $S-box$ as input that has the same functionality. It is also because one half can be expressed using the other half and X.

The verifications are exactly the same, I define the two mapping functions, one maps a pair of words to only a half, the other one use a single 6-bit message to produce the other half by exclusive-or operation with input XOR. Then by rewriting with BIJ_IFF_INV, $FINITE\_BIJ\_CARD$ and applying the definitions to the goal, BIJ relation and same cardinality verifications are done.

Differential Cryptanalysis using Probability Space The target set and the universal set is implemented, I can then implement probability space with the sample space, set of events and probability of event to help produce the probability. In HOL4, the theorem *prob_uniform_on_finite_set* states the prerequisites for a measure space to become a probability space which is suited for my usage. A measure space needs to meet the prerequisites that the measure space is a finite set and is not empty, the measurable set is the power set of measure space . Moreover, for each set in the power set, the measure of each set needs to be equal to the cardinality of each set divided by the cardinality of sample space. The space, measurable sets, measure of measure space are exactly the same with the components of probability space - sample space, set of events and the probability of event once measure space is proved also to be a probability space. From the prerequisites, we can obviously see that, the universal set word 6 can be the sample space. The sets of single 6-bit texts with input XOR X, output XOR Y and chosen S-boxes are the events that belong to the pow set of the universal set. The required probability p of X may cause Y can then be generated as the probability of events in that probability space.

```
⊢ FINITE (p_space p) ∧ p_space p ≠ ∅ ∧
  events p = POW (p_space p) ∧
  (∀s. s ∈ events p ⇒ prob p s = &CARD s / &CARD (p_space p)) ⇒
  prob_space p
```

To proving *prob_uniform_on_finite_set* given the prerequisites, I first convert the sample space, set of events and probability of event back to the space, measurable sets and measure by their definitions to match with the input p as a measure space. Then I verify that the measure space is not empty as it is finite. Then using the theorem *prob_on_finite_set*, I can expand the goal to three sub-goals. Once I proved that a measure space p is positive, additive and the probability for the space is 1, then p is a probability space.

```
⊢ FINITE (m_space p) ∧ measurable_sets p = POW (m_space p) ⇒
  (prob_space p  ⇔
   positive p ∧ prob p (p_space p) = 1 ∧ additive p)
```

The positivity of a measure space means the measure is 0 on empty set and greater than 0 for all other sets. By combining with the last prerequisite and definition of positivity, the sub-goal changes to prove the cardinality of empty set divided by the cardinality of space is 0 and the cardinality of all other sets divided by the cardinality of space is greater or equal to 0. This can be easily done by shown the cardinality of empty set is 0, all sets has cardinality >= 0 and the space is not empty. However, as calculations in measure space are in type *extreal* in HOL4, so built-in theorems in *extreal_Theory* are needed to help complete the proof. These theorems help do calculations, operations and comparisons for these numbers like the *extreal_lt_eq* theorem

```
⊢ Normal x < Normal y  ⇔  x < y
```

Then to show the probability for the space is 1, it is to show the cardinality of space out of the cardinality of itself is one. It is also quite obviously, but as the cardinality is in the type of *extreal* which contains positive and negative infinite, I need to ensure the cardinality of the space is not 0, positive infinite and negative infinite. It can be directly proved by the assumptions that space is a finite and non-empty set. But *extreal* definition is also needed to expand the operation & in the goal *extreal_of_num_def* to do the comparisons.

```
⊢ &n = Normal (&n)
```

At last, I need to prove the additivity of the measure space. It means for any two disjoint sets, if both of them and their union belong to the power set of the space, the addition of the measures of the two sets is equivalent to the measure of their union set. By applying the assumption, I want to solve the following goal:

$$\frac{\&\text{CARD }(s \cup t)}{\&\text{CARD }(m\_space\, p)} = \frac{\&\text{CARD }s}{\&\text{CARD }(m\_space\, p)} + \frac{\&\text{CARD }t}{\&\text{CARD }(m\_space\, p)} \quad (12)$$

I then want to show the cardinality of the union is equal to the sum of the two sets. By applying the tacitic *MATCH_MP_TAC* introduced earlier, I need to show the two sets are finite and disjoint from each other. The disjoint goal is directly proved. Use the theorem that for sets belong to the power set of a finite set, these sets are the subsets of that set, I can then prove these subsets are also finite. After some simple conversions, and use the theorem *div_add* with all the necessary prerequisites, the additivity is proved.

$\vdash e \in \texttt{POW}\ set \iff e \subseteq set$

$\vdash \texttt{FINITE}\ s \land t \subseteq s \Rightarrow \texttt{FINITE}\ t$

Overall, the measure space with the given prerequisites is also a probability space.

Now, I can build my measure spaces about differential cryptanalysis and the sets containing *XORed* texts. I can then use the implementations to generate the probability once I proved that they are probability spaces. a

7 Appendix: des_prop Theory

Parent Theories: probability, des

7.1 Definitions

[AllpairXor_def]

```
 ⊢ ∀X. AllpairXor X = {(x₁,x₂) | x₁ ⊕ x₂ = X}
```

[characterDES_def]

```
 ⊢ ∀X Y Yl.
     characterDES X Y Yl =
     (let
        XorR = GENLIST (λi. charapairDES X Yl i) (LENGTH Yl)
      in
        (X,XorR,Y))
```

[non_weak_keys_def]

```
 ⊢ non_weak_keys =
   [(0xB0B351C802C83DE0w,0x4739A2F04B7EAB28w);
    (0x5D460701328F2962w,0x9FE10D2E8C496143w);
    (0x4F4CAE37FD37C21Fw,0xB8C65D0FB48154D7w);
    (0xA2B9F8FECD70D69Dw,0x601EF2D173B69EBCw)]
```

[roundk_L]

```
 ⊢ (∀k. RK_L 0 k = FST (PC1 k)) ∧
   ∀n k.
     RK_L (SUC n) k =
     (let c = RK_L n k; r = EL n R_data in c ⇆ r)
```

[roundk_R]

```
 ⊢ (∀k. RK_R 0 k = SND (PC1 k)) ∧
   ∀n k.
     RK_R (SUC n) k =
     (let c = RK_R n k; r = EL n R_data in c ⇆ r)
```

[roundk_supp]

```
 ⊢ ∀n k. RK n k = (RK_L n k,RK_R n k)
```

[S_data_def]

```
 ⊢ S_data =
   [S8_data; S7_data; S6_data; S5_data; S4_data; S3_data;
    S2_data; S1_data]
```

[Semiwkey_def]

```
 ⊢ Semiwkey =
   [(0x1FE01FE01FE01FEw,0xFE01FE01FE01FE01w);
    (0x1FE01FE00EF10EF1w,0xE01FE01FF10EF10Ew);
    (0x1E001E001F101F1w,0xE001E001F101F101w);
    (0x1FFE1FFE0EFE0EFEw,0xFE1FFE1FFE0EFE0Ew);
    (0x11F011F010E010Ew,0x1F011F010E010E01w);
    (0xE0FEE0FEF1FEF1FEw,0xFEE0FEE0FEF1FEF1w)]
```

[splitXF_def]

```
 ⊢ ∀ Xe.
     splitXF Xe =
     [(5 >< 0) Xe; (11 >< 6) Xe; (17 >< 12) Xe; (23 >< 18) Xe;
      (29 >< 24) Xe; (35 >< 30) Xe; (41 >< 36) Xe;
      (47 >< 42) Xe]
```

[splitYF_def]

```
 ⊢ ∀ Ye.
     splitYF Ye =
     [(3 >< 0) Ye; (7 >< 4) Ye; (11 >< 8) Ye; (15 >< 12) Ye;
      (19 >< 16) Ye; (23 >< 20) Ye; (27 >< 24) Ye;
      (31 >< 28) Ye]
```

[trans1_def]

```
 ⊢ ∀ x1 x2. trans1 (x1,x2) = x1
```

[trans2_def]

```
 ⊢ ∀ x. trans2 x = (x,x)
```

[transF1_def]

```
 ⊢ ∀ x1 x2 x3 x4 x5 x6 x7 x8.
     transF1 (x1,x2,x3,x4,x5,x6,x7,x8) =
     x1 @@ x2 @@ x3 @@ x4 @@ x5 @@ x6 @@ x7 @@ x8
```

[transF2_def]

```
 ⊢ ∀ x. transF2 x =
       ((47 >< 42) x,(41 >< 36) x,(35 >< 30) x,(29 >< 24) x,
        (23 >< 18) x,(17 >< 12) x,(11 >< 6) x,(5 >< 0) x)
```

[transktoxF_def]

```
 ⊢ ∀ x k. transktoxF x k = k ⊕ x
```

[transxtokF_def]

```
 ⊢ ∀ x x′. transxtokF x x′ = x ⊕ x′
```

[w1trans1_def]

```
 ⊢ ∀ x. w1trans1 x = FST (Split (IP (desCore 8 (KS Wkey1 8) x)))
```

[w1trans2_def]

```
 ⊢ ∀ x. w1trans2 x = desCore 8 (KS Wkey1 8) (IIP (Join (x,x)))
```

[w2trans1_def]

```
 ⊢ ∀ x. w2trans1 x = FST (Split (IP (desCore 8 (KS Wkey2 8) x)))
```

[w2trans2_def]

```
⊢ ∀x. w2trans2 x = desCore 8 (KS Wkey2 8) (IIP (Join (x,x)))
```

[w3trans1_def]

```
⊢ ∀x. w3trans1 x = FST (Split (IP (desCore 8 (KS Wkey3 8) x)))
```

[w3trans2_def]

```
⊢ ∀x. w3trans2 x = desCore 8 (KS Wkey3 8) (IIP (Join (x,x)))
```

[w4trans1_def]

```
⊢ ∀x. w4trans1 x = FST (Split (IP (desCore 8 (KS Wkey4 8) x)))
```

[w4trans2_def]

```
⊢ ∀x. w4trans2 x = desCore 8 (KS Wkey4 8) (IIP (Join (x,x)))
```

[Wkey1_def]

```
⊢ Wkey1 = 0x101010101010101w
```

[Wkey2_def]

```
⊢ Wkey2 = 0xFEFEFEFEFEFEFEFEw
```

[Wkey3_def]

```
⊢ Wkey3 = 0xE0E0E0E0F1F1F1F1w
```

[Wkey4_def]

```
⊢ Wkey4 = 0x1F1F1F1F0E0E0E0Ew
```

[Wkey_def]

```
⊢ Wkey =
    [0x101010101010101w; 0xFEFEFEFEFEFEFEFEw;
     0xE0E0E0E0F1F1F1F1w; 0x1F1F1F1F0E0E0E0Ew]
```

[word32p_def]

```
⊢ word32p =
    (𝒰(:word32),POW 𝒰(:word32),(λs. &CARD s / &(2 ** 32)))
```

[word48p_def]

```
⊢ word48p =
    (𝒰(:word48),POW 𝒰(:word48),(λs. &CARD s / &(2 ** 48)))
```

[word6p_def]

```
⊢ word6p = (𝒰(:word6),POW 𝒰(:word6),(λs. &CARD s / &(2 ** 6)))
```

[word6x6_def]

```
⊢ word6x6 =
    (𝒰(:word6 × word6),POW 𝒰(:word6 × word6),
     (λs. &CARD s / &(2 ** 12)))
```

[Wtext_def]

```
⊢ ∀key.
      Wtext key =
      {x | ∃w. Split (IP (desCore 8 (KS key 8) x)) = (w,w)}
```

[Wtextlist_def]

```
⊢ Wtextlist = [Wtext1; Wtext2; Wtext3; Wtext4]
```

[wtrans1_def]

```
⊢ wtrans1 = [w1trans1; w2trans1; w3trans1; w4trans1]
```

[XcauseY_def]

$\vdash \forall X\ Y\ Sb.$
    XcauseY $X\ Y\ Sb$ = prob word6p $\{x \mid Sb\ x \oplus Sb\ (x \oplus X) = Y\}$

[XcauseYF'_def]

$\vdash \forall X\ Y.$
    XcauseYF' $X\ Y$ = prob word48p $\{x \mid \mathtt{S}\ x \oplus \mathtt{S}\ (x \oplus \mathtt{E}\ X) = Y\}$

[XcauseYF_def]

$\vdash \forall X\ Y.$
    XcauseYF $X\ Y$ =
    prob word32p
      $\{x \mid (\exists k.\ \mathtt{S}\ (\mathtt{E}\ x \oplus k) \oplus \mathtt{S}\ (\mathtt{E}\ x \oplus \mathtt{E}\ X \oplus k) = Y)\}$

[XcauseYFkey_def]

$\vdash \forall X\ Y\ x.$
    XcauseYFkey $X\ Y\ x$ =
    (**let**
       $x' = x \oplus \mathtt{E}\ X$
     in
       prob word48p $\{k \mid \mathtt{S}\ (x \oplus k) \oplus \mathtt{S}\ (x' \oplus k) = Y\}$)

[XcauseYFp_def]

$\vdash \forall X\ Y\ p.$
    XcauseYFp $X\ Y\ p \iff$
    prob word32p
      $\{x \mid (\exists k.\ \mathtt{S}\ (\mathtt{E}\ x \oplus k) \oplus \mathtt{S}\ (\mathtt{E}\ x \oplus \mathtt{E}\ X \oplus k) = Y)\}$ =
    p

[XcauseYp_def]

$\vdash \forall X\ Y\ Sb\ p.$
    XcauseYp $X\ Y\ Sb\ p \iff$
    prob word6p $\{x \mid Sb\ x \oplus Sb\ (x \oplus X) = Y\} = p$

[XpairF_def]

$\vdash \forall X\ Y.$
    XpairF $X\ Y$ =
    $\{(x_8,x_7,x_6,x_5,x_4,x_3,x_2,x_1) \mid$
     S8 $x_1 \oplus$ S8 $(x_1 \oplus$ (5 >< 0) (E X)) = (3 >< 0) $Y \wedge$
     S7 $x_2 \oplus$ S7 $(x_2 \oplus$ (11 >< 6) (E X)) = (7 >< 4) $Y \wedge$
     S6 $x_3 \oplus$ S6 $(x_3 \oplus$ (17 >< 12) (E X)) = (11 >< 8) $Y \wedge$
     S5 $x_4 \oplus$ S5 $(x_4 \oplus$ (23 >< 18) (E X)) = (15 >< 12) $Y \wedge$
     S4 $x_5 \oplus$ S4 $(x_5 \oplus$ (29 >< 24) (E X)) = (19 >< 16) $Y \wedge$
     S3 $x_6 \oplus$ S3 $(x_6 \oplus$ (35 >< 30) (E X)) = (23 >< 20) $Y \wedge$
     S2 $x_7 \oplus$ S2 $(x_7 \oplus$ (41 >< 36) (E X)) = (27 >< 24) $Y \wedge$
     S1 $x_8 \oplus$ S1 $(x_8 \oplus$ (47 >< 42) (E X)) = (31 >< 28) $Y\}$

7.2 Theorems

[AllpairXor_card]

$\vdash$ CARD (AllpairXor 0w) = 2 ** 6

[BIJ_F]

$\vdash \forall X\ Y.$ BIJ transF2 $\{x \mid \mathtt{S}\ x \oplus \mathtt{S}\ (x \oplus \mathtt{E}\ X) = Y\}$ (XpairF $X\ Y$)

[BIJ_for_weak_keys]

$\vdash \forall x.$ MEM x Wtextlist $\Rightarrow \exists f.$ BIJ $f\ x\ \mathcal{U}$(:word32)

```
[BIJ_for_weak_keys_explicit]
 ⊢ ∀i. i < 4 ⇒ BIJ (EL i wtrans1) (EL i Wtextlist) 𝒰(:word32)

[BIJ_ktox]
 ⊢ ∀X Y x.
     BIJ (transktoxF x) {k | S (k ⊕ x) ⊕ S (k ⊕ x ⊕ E X) = Y}
       {x | S x ⊕ S (x ⊕ E X) = Y}

[BIJ_wtext1]
 ⊢ BIJ w1trans1 Wtext1 𝒰(:word32)

[BIJ_wtext2]
 ⊢ BIJ w2trans1 Wtext2 𝒰(:word32)

[BIJ_wtext3]
 ⊢ BIJ w3trans1 Wtext3 𝒰(:word32)

[BIJ_wtext4]
 ⊢ BIJ w4trans1 Wtext4 𝒰(:word32)

[BIJ_XORL]
 ⊢ BIJ trans2 𝒰(:word6) (AllpairXor 0w)

[CARD_eqF]
 ⊢ ∀X Y. CARD {x | S x ⊕ S (x ⊕ E X) = Y} = CARD (XpairF X Y)

[CARD_kxeq]
 ⊢ ∀X Y x.
     CARD {k | S (k ⊕ x) ⊕ S (k ⊕ x ⊕ E X) = Y} =
     CARD {x | S x ⊕ S (x ⊕ E X) = Y}

[charapairDES_compute]
 ⊢ (∀Yl X. charapairDES X Yl 0 = ((31 >< 0) X,EL 0 Yl)) ∧
   (∀n Yl X.
      charapairDES X Yl <..num comp'n..> =
      if <..num comp'n..> = 1 then
        ((63 >< 32) X ⊕ EL 0 Yl,EL 1 Yl)
      else
        (let
           (Xin,Xout) =
             charapairDES X Yl (<..num comp'n..> − 1 − 1)
         in
           (Xin ⊕ EL (<..num comp'n..> − 1) Yl,
            EL <..num comp'n..> Yl))) ∧
   ∀n Yl X.
     charapairDES X Yl <..num comp'n..> =
     if <..num comp'n..> = 1 then
       ((63 >< 32) X ⊕ EL 0 Yl,EL 1 Yl)
     else
       (let
          (Xin,Xout) = charapairDES X Yl (<..num comp'n..> − 1)
        in
          (Xin ⊕ EL <..num comp'n..> Yl,EL <..num comp'n..> Yl))

[charapairDES_def]
 ⊢ (∀Yl X. charapairDES X Yl 0 = ((31 >< 0) X,EL 0 Yl)) ∧
   ∀n Yl X.
     charapairDES X Yl (SUC n) =
     if SUC n = 1 then ((63 >< 32) X ⊕ EL 0 Yl,EL 1 Yl)
     else
       (let
          (Xin,Xout) = charapairDES X Yl (n − 1)
        in
          (Xin ⊕ EL n Yl,EL (SUC n) Yl))
```

[charapairDES_ind]

$\vdash \forall P'.$
$(\forall X\ Yl.\ P'\ X\ Yl\ 0) \wedge$
$(\forall X\ Yl\ n.\ (\texttt{SUC}\ n \neq 1 \Rightarrow P'\ X\ Yl\ (n - 1)) \Rightarrow P'\ X\ Yl\ (\texttt{SUC}\ n)) \Rightarrow$
$\forall v\ v_1\ v_2.\ P'\ v\ v_1\ v_2$

[compl_E]

$\vdash \forall m.\ \texttt{E}\ (\neg m) = \neg\texttt{E}\ m$

[compl_extract_1]

$\vdash \forall m.\ (63 \texttt{ >< } 32)\ (\neg m) = \neg(63 \texttt{ >< } 32)\ m$

[compl_extract_2]

$\vdash \forall m.\ (31 \texttt{ >< } 0)\ (\neg m) = \neg(31 \texttt{ >< } 0)\ m$

[compl_IIP]

$\vdash \forall m.\ \texttt{IIP}\ (\neg m) = \neg\texttt{IIP}\ m$

[compl_IP]

$\vdash \forall m.\ \texttt{IP}\ (\neg m) = \neg\texttt{IP}\ m$

[compl_join]

$\vdash \forall m\ n.\ \texttt{Join}\ (\neg m,\neg n) = \neg\texttt{Join}\ (m,n)$

[compl_RK_L]

$\vdash \forall n\ k.\ 17 > n \Rightarrow \texttt{RK\_L}\ n\ (\neg k) = \neg\texttt{RK\_L}\ n\ k$

[compl_RK_R]

$\vdash \forall n\ k.\ 17 > n \Rightarrow \texttt{RK\_R}\ n\ (\neg k) = \neg\texttt{RK\_R}\ n\ k$

[comple_PC2]

$\vdash \forall a\ b.\ \texttt{PC2}\ (\neg a,\neg b) = \neg\texttt{PC2}\ (a,b)$

[comple_property]

$\vdash \forall k\ m\ n.$
$0 < n \wedge n < 17 \wedge \texttt{DES}\ n\ k = (\mathit{encrypt},\mathit{decrypt}) \wedge$
$\texttt{DES}\ n\ (\neg k) = (\mathit{encrypt}',\mathit{decrypt}') \Rightarrow$
$\neg\mathit{encrypt}\ m = \mathit{encrypt}'\ (\neg m)$

[convert_RK]

$\vdash \forall k\ n.\ \texttt{RoundKey}\ n\ k = \texttt{REVERSE}\ (\texttt{GENLIST}\ (\lambda i.\ \texttt{RK}\ i\ k)\ (\texttt{SUC}\ n))$

[DES_fp_non_weak_keys]

$\vdash \forall i.\ i < \texttt{LENGTH non\_weak\_keys} \Rightarrow$
(**let**
$(\mathit{key},\mathit{plaintext}) = \texttt{EL}\ i\ \texttt{non\_weak\_keys};$
$(\mathit{encrypt},\mathit{decrypt}) = \texttt{FullDES}\ \mathit{key}$
in
$\mathit{encrypt}\ \mathit{plaintext} = \mathit{plaintext}$)

[DES_weak_fp_card]

$\vdash \forall x.\ \texttt{MEM}\ x\ \texttt{Wtextlist} \Rightarrow \texttt{CARD}\ x = 2\ \texttt{**}\ 32$

[F_convert]

$\vdash \forall X\ Y.$
$\texttt{XpairF}\ X\ Y =$
$\{x \mid \texttt{S1}\ x \oplus \texttt{S1}\ (x \oplus (47 \texttt{ >< } 42)\ (\texttt{E}\ X)) = (31 \texttt{ >< } 28)\ Y\} \times$
$\{x \mid \texttt{S2}\ x \oplus \texttt{S2}\ (x \oplus (41 \texttt{ >< } 36)\ (\texttt{E}\ X)) = (27 \texttt{ >< } 24)\ Y\} \times$
$\{x \mid \texttt{S3}\ x \oplus \texttt{S3}\ (x \oplus (35 \texttt{ >< } 30)\ (\texttt{E}\ X)) = (23 \texttt{ >< } 20)\ Y\} \times$
$\{x \mid \texttt{S4}\ x \oplus \texttt{S4}\ (x \oplus (29 \texttt{ >< } 24)\ (\texttt{E}\ X)) = (19 \texttt{ >< } 16)\ Y\} \times$
$\{x \mid \texttt{S5}\ x \oplus \texttt{S5}\ (x \oplus (23 \texttt{ >< } 18)\ (\texttt{E}\ X)) = (15 \texttt{ >< } 12)\ Y\} \times$
$\{x \mid \texttt{S6}\ x \oplus \texttt{S6}\ (x \oplus (17 \texttt{ >< } 12)\ (\texttt{E}\ X)) = (11 \texttt{ >< } 8)\ Y\} \times$
$\{x \mid \texttt{S7}\ x \oplus \texttt{S7}\ (x \oplus (11 \texttt{ >< } 6)\ (\texttt{E}\ X)) = (7 \texttt{ >< } 4)\ Y\} \times$
$\{x \mid \texttt{S8}\ x \oplus \texttt{S8}\ (x \oplus (5 \texttt{ >< } 0)\ (\texttt{E}\ X)) = (3 \texttt{ >< } 0)\ Y\}$

[prob_space_word32p]

```
 ⊢ prob_space word32p
```

[prob_space_word48p]

```
 ⊢ prob_space word48p
```

[prob_space_word6p]

```
 ⊢ prob_space word6p
```

[prob_space_word6x6]

```
 ⊢ prob_space word6x6
```

[prob_uniform_on_finite_set]

```
 ⊢ ∀ p. FINITE (p_space p) ∧ p_space p ≠ { } ∧
       events p = POW (p_space p) ∧
       (∀ s. s ∈ events p ⇒
            prob p s = &CARD s / &CARD (p_space p)) ⇒
       prob_space p
```

[roundk_L_compute]

```
 ⊢ (∀ k. RK_L 0 k = FST (PC1 k)) ∧
   (∀ n k.
      RK_L <..num comp'n..> k =
      (let
         c = RK_L (<..num comp'n..> − 1) k;
         r = EL (<..num comp'n..> − 1) R_data
       in
         c ⇆ r)) ∧
   ∀ n k.
     RK_L <..num comp'n..> k =
     (let
        c = RK_L <..num comp'n..> k;
        r = EL <..num comp'n..> R_data
      in
        c ⇆ r)
```

[roundk_R_compute]

```
 ⊢ (∀ k. RK_R 0 k = SND (PC1 k)) ∧
   (∀ n k.
      RK_R <..num comp'n..> k =
      (let
         c = RK_R (<..num comp'n..> − 1) k;
         r = EL (<..num comp'n..> − 1) R_data
       in
         c ⇆ r)) ∧
   ∀ n k.
     RK_R <..num comp'n..> k =
     (let
        c = RK_R <..num comp'n..> k;
        r = EL <..num comp'n..> R_data
      in
        c ⇆ r)
```

[semiK_proper1]

```
 ⊢ ∀ plaintext pair.
     MEM pair Semiwkey ∧ pair = (s1,s2) ∧
     FullDES s1 = (encrypt,decrypt) ∧
     FullDES s2 = (encrypt',decrypt') ⇒
     encrypt (encrypt' plaintext) = plaintext
```

[semiK_proper2]

$\vdash \forall\, \mathit{plaintext}\ \mathit{pair}.$
$\quad \mathtt{MEM}\ \mathit{pair}\ \mathtt{Semiwkey} \land \mathit{pair} = (s_1, s_2) \land$
$\quad \mathtt{FullDES}\ s_1 = (\mathit{encrypt}, \mathit{decrypt}) \land$
$\quad \mathtt{FullDES}\ s_2 = (\mathit{encrypt}', \mathit{decrypt}') \Rightarrow$
$\quad \mathit{encrypt}'\ (\mathit{encrypt}\ \mathit{plaintext}) = \mathit{plaintext}$

[text_num]

$\vdash \forall x.\ \mathtt{MEM}\ x\ \mathtt{Wtextlist} \Rightarrow \mathtt{CARD}\ x = \mathtt{CARD}\ \mathcal{U}(\mathtt{:word32})$

[weakK1_proper2]

$\vdash \forall x.\ x \in \mathtt{Wtext1} \land \mathtt{FullDES}\ \mathtt{Wkey1} = (\mathit{encrypt}, \mathit{decrypt}) \Rightarrow$
$\quad \mathit{encrypt}\ x = x$

[weakK2_proper2]

$\vdash \forall x.\ x \in \mathtt{Wtext2} \land \mathtt{FullDES}\ \mathtt{Wkey2} = (\mathit{encrypt}, \mathit{decrypt}) \Rightarrow$
$\quad \mathit{encrypt}\ x = x$

[weakK3_proper2]

$\vdash \forall x.\ x \in \mathtt{Wtext3} \land \mathtt{FullDES}\ \mathtt{Wkey3} = (\mathit{encrypt}, \mathit{decrypt}) \Rightarrow$
$\quad \mathit{encrypt}\ x = x$

[weakK4_proper2]

$\vdash \forall x.\ x \in \mathtt{Wtext4} \land \mathtt{FullDES}\ \mathtt{Wkey4} = (\mathit{encrypt}, \mathit{decrypt}) \Rightarrow$
$\quad \mathit{encrypt}\ x = x$

[weakK_proper]

$\vdash \forall k\ \mathit{plaintext}.$
$\quad \mathtt{MEM}\ k\ \mathtt{Wkey} \land \mathtt{FullDES}\ k = (\mathit{encrypt}, \mathit{decrypt}) \Rightarrow$
$\quad \mathit{encrypt}\ (\mathit{encrypt}\ \mathit{plaintext}) = \mathit{plaintext}$

[weakK_sup]

$\vdash \forall n\ k.$
$\quad \mathtt{MEM}\ k\ \mathtt{Wkey} \land 0 \le n \land n \le 8 \land$
$\quad \mathtt{Split}\ (\mathtt{IP}\ (\mathtt{desCore}\ 8\ (\mathtt{KS}\ k\ 8)\ x)) = (w, w) \Rightarrow$
$\quad \mathtt{Round}\ (8 - n)\ (\mathtt{KS}\ k\ 16)\ (\mathtt{Split}\ (\mathtt{IP}\ x)) =$
$\quad \mathtt{Swap}\ (\mathtt{Round}\ (8 + n)\ (\mathtt{KS}\ k\ 16)\ (\mathtt{Split}\ (\mathtt{IP}\ x)))$

[wkey1_equal]

$\vdash \forall x\ n\ k.$
$\quad \mathtt{MEM}\ k\ \mathtt{Wkey} \land n \le 8 \Rightarrow$
$\quad \mathtt{Round}\ n\ (\mathtt{KS}\ k\ 8)\ (\mathtt{Split}\ x) = \mathtt{Round}\ n\ (\mathtt{KS}\ k\ 16)\ (\mathtt{Split}\ x)$

[word6p_convert]

$\vdash \{x \mid \mathtt{S1}\ x \oplus \mathtt{S1}\ (x \oplus \mathtt{52w}) = \mathtt{4w}\} = \{\mathtt{19w};\ \mathtt{39w}\}$

[Wtext1_def]

$\vdash \mathtt{Wtext1} =$
$\quad \{x \mid \exists w.\ \mathtt{Split}\ (\mathtt{IP}\ (\mathtt{desCore}\ 8\ (\mathtt{KS}\ \mathtt{Wkey1}\ 8)\ x)) = (w, w)\}$

[Wtext2_def]

$\vdash \mathtt{Wtext2} =$
$\quad \{x \mid \exists w.\ \mathtt{Split}\ (\mathtt{IP}\ (\mathtt{desCore}\ 8\ (\mathtt{KS}\ \mathtt{Wkey2}\ 8)\ x)) = (w, w)\}$

[Wtext3_def]

$\vdash \mathtt{Wtext3} =$
$\quad \{x \mid \exists w.\ \mathtt{Split}\ (\mathtt{IP}\ (\mathtt{desCore}\ 8\ (\mathtt{KS}\ \mathtt{Wkey3}\ 8)\ x)) = (w, w)\}$

[Wtext4_def]

```
⊢ Wtext4 =
  {x | ∃w. Split (IP (desCore 8 (KS Wkey4 8) x)) = (w,w)}
```

[XcauseYF_convert]

```
⊢ ∀X Y x. XcauseYFkey X Y x = XcauseYF' X Y
```

[XcauseYFp_eq]

```
⊢ ∀X Y.
     Xe = E X ∧ Xl = splitXF Xe ∧ Yl = splitYF Y ⇒
     XcauseYF' X Y =
     ∏ (λi. XcauseY (EL i Xl) (EL i Yl) (SBox (EL i S_data)))
       (count 8)
```

[XcauseYp_test]

```
⊢ XcauseYp 52w 4w S1 (2 / 64)
```

[xor_E]

```
⊢ ∀x₁ x₂. E x₁ ⊕ E x₂ = E (x₁ ⊕ x₂)
```

[xor_P]

```
⊢ ∀x₁ x₂. P x₁ ⊕ P x₂ = P (x₁ ⊕ x₂)
```

[xor_S1]

```
⊢ ∃x₁ x₂. S1 x₁ ⊕ S1 x₂ ≠ S1 (x₁ ⊕ x₂)
```

8 Appendix: rc5 Theory

Parent Theories: words, sorting

8.1 Definitions

[Join64'_def]

```
⊢ ∀b. Join64' b = (let (u,v) = b in u @@ v)
```

[Join64_def]

```
⊢ ∀u v. Join64 (u,v) = u @@ v
```

[keys_def]

```
⊢ (∀r k.
      keys 0 r k =
      (let
         Lk = Lkeys k;
         Sk = Skeys r;
         A = EL 0 Sk;
         B = EL 0 Lk
       in
         (A,B,Lk,Sk,0,0))) ∧
  ∀n r k.
     keys (SUC n) r k =
     (let
        (A,B,Lk,Sk,i,j) = keys n r k;
        Anew = (EL i Sk + A + B) ⇆ 3;
        Bnew = (EL i Lk + Anew + B) ⇆ w2n (Anew + B);
        Sknew =
          GENLIST
            (λm.
                 if m = i then (EL i Sk + A + B) ⇆ 3
                 else EL m Sk) (2 × (r + 1));
        Lknew =
          GENLIST
            (λ m.
                 if m = j then
                   (EL j Lk + Anew + B) ⇆ w2n (Anew + B)
                 else EL m Lk) 2;
        inew = (i + 1) MOD (2 × (r + 1));
        jnew = (j + 1) MOD 2
      in
        (Anew,Bnew,Lknew,Sknew,inew,jnew))
```

[keysIni_def]

```
 ⊢ ∀ k. keysIni k =
       [(7 >< 0) k; (15 >< 8) k; (23 >< 16) k; (31 >< 24) k;
        (39 >< 32) k; (47 >< 40) k; (55 >< 48) k; (63 >< 56) k]
```

[lenKeyw_def]

```
 ⊢ lenKeyw = 2
```

[lenKinbyt_def]

```
 ⊢ lenKinbyt = 8
```

[lenWinbyt_def]

```
 ⊢ lenWinbyt = 4
```

[Lkeys_def]

```
 ⊢ ∀ k. Lkeys k = LkeysSup k 7
```

[LkeysIni_def]

```
 ⊢ LkeysIni = [0w; 0w]
```

[LkeysSup_def]

```
 ⊢ (∀ k. LkeysSup k 0 =
        (let
           ks = LkeysIni;
           keys = keysIni k
         in
           GENLIST
             (λ m.
                  if m = 7 DIV 4 then EL m ks ⇆ 8 + EL 7 keys
                  else EL m ks) 2)) ∧
   ∀ k r.
     LkeysSup k (SUC r) =
     (let
        ks = LkeysSup k r;
        keys = keysIni k;
        i = 7 − SUC r
      in
        GENLIST
          (λ m.
               if m = i DIV 4 then EL m ks ⇆ 8 + EL i keys
               else EL m ks) 2)
```

[P32_data]

```
 ⊢ P32_data = 0xB7E15163w
```

[Q32_data]

```
 ⊢ Q32_data = 0x9E3779B9w
```

[rc5keys_def]
```
⊢ ∀ $r$ $k$.
    rc5keys $r$ $k$ =
    (let $n$ = $3 \times$ MAX $(2 \times (r + 1))$ 2 in keys $n$ $r$ $k$)
```
[RoundDe'_def]
```
⊢ (∀ $ki$ $ki_2$ $b$. RoundDe' 0 $ki$ $ki_2$ $b$ = $b$) ∧
  ∀ $n$ $ki$ $ki_2$ $b$.
    RoundDe' (SUC $n$) $ki$ $ki_2$ $b$ =
    (let $b'$ = RoundDe' $n$ $ki$ $ki_2$ $b$ in RoundDeSg $b'$ $ki$ $ki_2$)
```
[RoundDe64_def]
```
⊢ ∀ $n$ $w$ $k$.
    RoundDe64 $n$ $w$ $k$ =
    (let
       $(w_1, w_2)$ = Split64 $w$;
       $(A, B, Lk, Sk, i, j)$ = rc5keys $n$ $k$;
       $(w_1', w_2')$ = RoundDe $n$ $Sk$ $(w_1, w_2)$;
       $k_0$ = EL 0 $Sk$;
       $k_1$ = EL 1 $Sk$
     in
       Join64' $(w_1' - k_0, w_2' - k_1)$)
```
[RoundDeSg_def]
```
⊢ ∀ $b$ $ki$ $ki_2$.
    RoundDeSg $b$ $ki$ $ki_2$ =
    (let
       $(w_1, w_2)$ = $b$;
       $B$ = $(w_2 - ki_2) \rightleftarrows$ w2n $w_1 \oplus w_1$;
       $A$ = $(w_1 - ki) \rightleftarrows$ w2n $B \oplus B$
     in
       $(A, B)$)
```
[RoundEn'_def]
```
⊢ (∀ $ki$ $ki_2$ $b$. RoundEn' 0 $ki$ $ki_2$ $b$ = $b$) ∧
  ∀ $n$ $ki$ $ki_2$ $b$.
    RoundEn' (SUC $n$) $ki$ $ki_2$ $b$ =
    (let $b'$ = RoundEn' $n$ $ki$ $ki_2$ $b$ in RoundEnSg $b'$ $ki$ $ki_2$)
```
[RoundEn64_def]
```
⊢ ∀ $n$ $w$ $k$.
    RoundEn64 $n$ $w$ $k$ =
    (let
       $(w_1, w_2)$ = Split64 $w$;
       $(A, B, Lk, Sk, i, j)$ = rc5keys $n$ $k$;
       $k_0$ = EL 0 $Sk$;
       $k_1$ = EL 1 $Sk$
     in
       Join64' (RoundEn $n$ $(w_1 + k_0)$ $(w_2 + k_1)$ $Sk$))
```
[RoundEn_def]
```
⊢ (∀ $w_1$ $w_2$ $ks$.
     RoundEn 0 $w_1$ $w_2$ $ks$ =
     (let $s_0$ = EL 0 $ks$; $s_1$ = EL 1 $ks$ in $(w_1, w_2)$)) ∧
  ∀ $n$ $w_1$ $w_2$ $ks$.
    RoundEn (SUC $n$) $w_1$ $w_2$ $ks$ =
    (let
       $(w_1', w_2')$ = RoundEn $n$ $w_1$ $w_2$ $ks$;
       $ki$ = EL $(2 \times$ SUC $n)$ $ks$;
       $ki_2$ = EL $(2 \times$ SUC $n + 1)$ $ks$;
       $A$ = $(w_1' \oplus w_2') \leftrightarrows$ w2n $w_2' + ki$;
       $B$ = $(w_2' \oplus A) \leftrightarrows$ w2n $A + ki_2$
     in
       $(A, B)$)
```

[RoundEnSg_def]

```
⊢ ∀b ki ki₂.
     RoundEnSg b ki ki₂ =
     (let
        (w₁,w₂) = b;
        A = (w₁ ⊕ w₂) ⇆ w2n w₂ + ki;
        B = (w₂ ⊕ A) ⇆ w2n A + ki₂
      in
        (A,B))
```

[Skeys_def]

```
⊢ ∀r. Skeys r = REVERSE (SkeysT32 32 (2 × (r + 1) − 1))
```

[SkeysT32_def]

```
⊢ (∀l. SkeysT32 l 0 = [P32_data]) ∧
   ∀l t.
     SkeysT32 l (SUC t) =
     (let ks = SkeysT32 l t; key = HD ks in key + Q32_data::ks)
```

[Split64_def]

```
⊢ ∀w. Split64 w = ((63 >< 32) w,(31 >< 0) w)
```

8.2 Theorems

[half_messageDe_def]

```
⊢ ∀w₂ w₁ n ks.
     half_messageDe w₁ w₂ ks n =
     (let
        ki = EL (n − 2) (REVERSE ks);
        ki₂ = EL (n − 2) (REVERSE ks)
      in
        if n = 0 then w₂
        else if n = 1 then w₁
        else
          (half_messageDe w₁ w₂ ks (n − 2) − ki) ⇄
          w2n (half_messageDe w₁ w₂ ks (n − 1)) ⊕
          half_messageDe w₁ w₂ ks (n − 1))
```

[half_messageDe_ind]

```
⊢ ∀P. (∀w₁ w₂ ks n.
          (∀ki ki₂.
             ki = EL (n − 2) (REVERSE ks) ∧
             ki₂ = EL (n − 2) (REVERSE ks) ∧ n ≠ 0 ∧ n ≠ 1 ⇒
             P w₁ w₂ ks (n − 1)) ∧
          (∀ki ki₂.
             ki = EL (n − 2) (REVERSE ks) ∧
             ki₂ = EL (n − 2) (REVERSE ks) ∧ n ≠ 0 ∧ n ≠ 1 ⇒
             P w₁ w₂ ks (n − 2)) ⇒
          P w₁ w₂ ks n) ⇒
       ∀v v₁ v₂ v₃. P v v₁ v₂ v₃
```

[half_messageEn_def]

```
⊢ ∀w₂ w₁ n ks.
     half_messageEn w₁ w₂ ks n =
     (let
        ki = EL n ks;
        ki₂ = EL (n − 1) ks
      in
        if n = 0 then w₁
        else if n = 1 then w₂
```

```
          else
            (half_messageEn w1 w2 ks (n − 2) ⊕
             half_messageEn w1 w2 ks (n − 1)) ⇆
            w2n (half_messageEn w1 w2 ks (n − 1)) + ki)
```

[half_messageEn_ind]

```
 ⊢ ∀P. (∀w1 w2 ks n.
          (∀ki ki2.
             ki = EL n ks ∧ ki2 = EL (n − 1) ks ∧ n ≠ 0 ∧ n ≠ 1 ⇒
             P w1 w2 ks (n − 1)) ∧
          (∀ki ki2.
             ki = EL n ks ∧ ki2 = EL (n − 1) ks ∧ n ≠ 0 ∧ n ≠ 1 ⇒
             P w1 w2 ks (n − 2)) ⇒
          P w1 w2 ks n) ⇒
       ∀v v1 v2 v3. P v v1 v2 v3
```

[Join64'_Split64]

```
 ⊢ ∀w. Join64' (Split64 w) = w
```

[Join64_Split64]

```
 ⊢ ∀w. Join64 (Split64 w) = w
```

[keys_compute]

```
 ⊢ (∀r k.
      keys 0 r k =
      (let
         Lk = Lkeys k;
         Sk = Skeys r;
         A = EL 0 Sk;
         B = EL 0 Lk
       in
         (A,B,Lk,Sk,0,0))) ∧
   (∀n r k.
      keys <..num comp'n..> r k =
      (let
         (A,B,Lk,Sk,i,j) = keys (<..num comp'n..> − 1) r k;
         Anew = (EL i Sk + A + B) ⇆ 3;
         Bnew = (EL i Lk + Anew + B) ⇆ w2n (Anew + B);
         Sknew =
           GENLIST
             (λm.
                  if m = i then (EL i Sk + A + B) ⇆ 3
                  else EL m Sk) (2 × (r + 1));
         Lknew =
           GENLIST
             (λm.
                  if m = j then
                    (EL j Lk + Anew + B) ⇆ w2n (Anew + B)
                  else EL m Lk) 2;
         inew = (i + 1) MOD (2 × (r + 1));
         jnew = (j + 1) MOD 2
       in
         (Anew,Bnew,Lknew,Sknew,inew,jnew))) ∧
   ∀n r k.
     keys <..num comp'n..> r k =
     (let
        (A,B,Lk,Sk,i,j) = keys <..num comp'n..> r k;
        Anew = (EL i Sk + A + B) ⇆ 3;
        Bnew = (EL i Lk + Anew + B) ⇆ w2n (Anew + B);
        Sknew =
          GENLIST
```

```
            (λ m.
                 if m = i then (EL i Sk + A + B) ⇆ 3
                 else EL m Sk) (2 × (r + 1));
        Lknew =
          GENLIST
            (λ m.
                 if m = j then
                   (EL j Lk + Anew + B) ⇆ w2n (Anew + B)
                 else EL m Lk) 2;
        inew = (i + 1) MOD (2 × (r + 1));
        jnew = (j + 1) MOD 2
      in
        (Anew,Bnew,Lknew,Sknew,inew,jnew))
```

[LENGTH_key_Lkeys]

```
 ⊢ ∀n r k. (A,B,Lk,Sk,i,j) = keys n r k ⇒ LENGTH Lk = 2
```

[LENGTH_key_Skeys]

```
 ⊢ ∀n r k.
     (A,B,Lk,Sk,i,j) = keys n r k ⇒ LENGTH Sk = 2 × (r + 1)
```

[LENGTH_Lkeys]

```
 ⊢ ∀k. LENGTH (Lkeys k) = 2
```

[LENGTH_LkeysSup]

```
 ⊢ ∀k r. LENGTH (LkeysSup k r) = 2
```

[LENGTH_Skeys]

```
 ⊢ ∀n. LENGTH (Skeys n) = 2 × (n + 1)
```

[LkeysSup_compute]

```
 ⊢ (∀k. LkeysSup k 0 =
        (let
           ks = LkeysIni;
           keys = keysIni k
         in
           GENLIST
             (λ m.
                  if m = 7 DIV 4 then EL m ks ⇆ 8 + EL 7 keys
                  else EL m ks) 2)) ∧
   (∀k r.
      LkeysSup k <..num comp'n..> =
      (let
         ks = LkeysSup k (<..num comp'n..> − 1);
         keys = keysIni k;
         i = 7 − <..num comp'n..>
       in
         GENLIST
           (λ m.
                if m = i DIV 4 then EL m ks ⇆ 8 + EL i keys
                else EL m ks) 2)) ∧
   ∀k r.
     LkeysSup k <..num comp'n..> =
     (let
        ks = LkeysSup k <..num comp'n..> ;
        keys = keysIni k;
        i = 7 − <..num comp'n..>
      in
        GENLIST
          (λ m.
               if m = i DIV 4 then EL m ks ⇆ 8 + EL i keys
               else EL m ks) 2)
```

[RoundDe'_comm]

```
 ⊢ ∀n b ks ki ki2.
     RoundDeSg (RoundDe' n ki ki2 b) ki ki2 =
     RoundDe' n ki ki2 (RoundDeSg b ki ki2)
```

[RoundDe'_compute]

```
 ⊢ (∀ki ki2 b. RoundDe' 0 ki ki2 b = b) ∧
   (∀n ki ki2 b.
      RoundDe' <..num comp'n..> ki ki2 b =
      (let
         b' = RoundDe' (<..num comp'n..> − 1) ki ki2 b
       in
         RoundDeSg b' ki ki2)) ∧
   ∀n ki ki2 b.
     RoundDe' <..num comp'n..> ki ki2 b =
     (let
        b' = RoundDe' <..num comp'n..> ki ki2 b
      in
        RoundDeSg b' ki ki2)
```

[RoundDe64_alt_half_messageDe]

```
 ⊢ ∀w k r.
     (w1,w2) = Split64 w ∧ (A,B,Lk,Sk,i,j) = rc5keys r k ∧
     k0 = EL 0 Sk ∧ k1 = EL 1 Sk ⇒
     RoundDe64 r w k =
     Join64'
       (half_messageDe w1 w2 Sk (2 × r + 1) − k0,
        half_messageDe w1 w2 Sk (2 × r) − k1)
```

[RoundDe_alt_half_messageDe]

```
 ⊢ ∀w1 w2 ks n.
     RoundDe n ks (w1,w2) =
     (half_messageDe w1 w2 ks (2 × n + 1),
      half_messageDe w1 w2 ks (2 × n))
```

[RoundDe_compute]

```
 ⊢ (∀w2 w1 ks.
      RoundDe 0 ks (w1,w2) =
      (let s1 = EL 1 ks; s0 = EL 0 ks in (w1,w2))) ∧
   (∀w2 w1 n ks.
      RoundDe <..num comp'n..> ks (w1,w2) =
      (let
         (w1',w2') = RoundDe (<..num comp'n..> − 1) ks (w1,w2);
         ki = EL (2 × <..num comp'n..> − 2) (REVERSE ks);
         ki2 = EL (2 × <..num comp'n..> − 1) (REVERSE ks);
         B = (w2' − ki) ⇄ w2n w1' ⊕ w1';
         A = (w1' − ki2) ⇄ w2n B ⊕ B
       in
         (A,B))) ∧
   ∀w2 w1 n ks.
     RoundDe <..num comp'n..> ks (w1,w2) =
     (let
        (w1',w2') = RoundDe <..num comp'n..> ks (w1,w2);
        ki = EL (2 × <..num comp'n..> − 2) (REVERSE ks);
        ki2 = EL (2 × <..num comp'n..> − 1) (REVERSE ks);
        B = (w2' − ki) ⇄ w2n w1' ⊕ w1';
        A = (w1' − ki2) ⇄ w2n B ⊕ B
      in
        (A,B))
```

[RoundDe_def]

```
 ⊢ (∀w2 w1 ks.
      RoundDe 0 ks (w1,w2) =
      (let s1 = EL 1 ks; s0 = EL 0 ks in (w1,w2))) ∧
   ∀w2 w1 n ks.
     RoundDe (SUC n) ks (w1,w2) =
     (let
        (w1',w2') = RoundDe n ks (w1,w2);
        ki = EL (2 × SUC n − 2) (REVERSE ks);
        ki2 = EL (2 × SUC n − 1) (REVERSE ks);
        B = (w2' − ki) ⇄ w2n w1' ⊕ w1';
        A = (w1' − ki2) ⇄ w2n B ⊕ B
      in
        (A,B))
```

[RoundDe_EnSg]

```
 ⊢ ∀b ki ki2. RoundDeSg (RoundEnSg b ki ki2) ki ki2 = b
```

[RoundDe_ind]

```
 ⊢ ∀P. (∀ks w1 w2. P 0 ks (w1,w2)) ∧
       (∀n ks w1 w2. P n ks (w1,w2) ⇒ P (SUC n) ks (w1,w2)) ⇒
       ∀v v1 v2 v3. P v v1 (v2,v3)
```

[RoundEe'_De']

```
 ⊢ ∀n b ki ki2. RoundDe' n ki ki2 (RoundEn' n ki ki2 b) = b
```

[RoundEn'_comm]

```
 ⊢ ∀n b ks ki ki2.
     RoundEnSg (RoundEn' n ki ki2 b) ki ki2 =
     RoundEn' n ki ki2 (RoundEnSg b ki ki2)
```

[RoundEn'_compute]

```
 ⊢ (∀ki ki2 b. RoundEn' 0 ki ki2 b = b) ∧
   (∀n ki ki2 b.
      RoundEn' <..num comp'n..> ki ki2 b =
      (let
         b' = RoundEn' (<..num comp'n..> − 1) ki ki2 b
       in
         RoundEnSg b' ki ki2)) ∧
   ∀n ki ki2 b.
     RoundEn' <..num comp'n..> ki ki2 b =
     (let
        b' = RoundEn' <..num comp'n..> ki ki2 b
      in
        RoundEnSg b' ki ki2)
```

[RoundEn64_alt_half_messageEn]

```
 ⊢ ∀w k r.
     (w1,w2) = Split64 w ∧ (A,B,Lk,Sk,i,j) = rc5keys r k ∧
     k0 = EL 0 Sk ∧ k1 = EL 1 Sk ⇒
     RoundEn64 r w k =
     Join64'
       (half_messageEn (w1 + k0) (w2 + k1) Sk (2 × r),
        half_messageEn (w1 + k0) (w2 + k1) Sk (2 × r + 1))
```

[RoundEn_alt_half_messageEn]

```
 ⊢ ∀w1 w2 ks n.
     RoundEn n w1 w2 ks =
     (half_messageEn w1 w2 ks (2 × n),
      half_messageEn w1 w2 ks (2 × n + 1))
```

[RoundEn_compute]

```
 ⊢ (∀ $w_1$ $w_2$ ks.
      RoundEn 0 $w_1$ $w_2$ ks =
      (let $s_0$ = EL 0 ks; $s_1$ = EL 1 ks in ($w_1$,$w_2$))) ∧
   (∀ n $w_1$ $w_2$ ks.
      RoundEn <..num comp'n..> $w_1$ $w_2$ ks =
      (let
         ($w_1'$,$w_2'$) = RoundEn (<..num comp'n..> − 1) $w_1$ $w_2$ ks;
         ki = EL (2 × <..num comp'n..> ) ks;
         $ki_2$ = EL (2 × <..num comp'n..> + 1) ks;
         A = ($w_1'$ ⊕ $w_2'$) ⇆ w2n $w_2'$ + ki;
         B = ($w_2'$ ⊕ A) ⇆ w2n A + $ki_2$
       in
         (A,B))) ∧
   ∀ n $w_1$ $w_2$ ks.
     RoundEn <..num comp'n..> $w_1$ $w_2$ ks =
     (let
        ($w_1'$,$w_2'$) = RoundEn <..num comp'n..> $w_1$ $w_2$ ks;
        ki = EL (2 × <..num comp'n..> ) ks;
        $ki_2$ = EL (2 × <..num comp'n..> + 1) ks;
        A = ($w_1'$ ⊕ $w_2'$) ⇆ w2n $w_2'$ + ki;
        B = ($w_2'$ ⊕ A) ⇆ w2n A + $ki_2$
      in
        (A,B))
```

[SkeysT32_compute]

```
 ⊢ (∀ l. SkeysT32 l 0 = [P32_data]) ∧
   (∀ l t.
      SkeysT32 l <..num comp'n..> =
      (let
         ks = SkeysT32 l (<..num comp'n..> − 1);
         key = HD ks
       in
         key + Q32_data::ks)) ∧
   ∀ l t.
     SkeysT32 l <..num comp'n..> =
     (let
        ks = SkeysT32 l <..num comp'n..> ;
        key = HD ks
      in
        key + Q32_data::ks)
```

[Split64_Join64]

```
 ⊢ ∀ u v. Split64 (Join64 (u,v)) = (u,v)
```

[Split64_Join64']

```
 ⊢ ∀ b. Split64 (Join64' b) = b
```