RC4 State and Its Applications

Qian Yu and Chang N. Zhang
Department of Computer Science, University of Regina
3737 Wascana Parkway, Regina SK, S4S 0A2, Canada
{yu209, zhang}@cs.uregina.ca

***Abstract* - In this paper, we define the RC4 state and introduce its forward and backward property. Based on the RC4 state and its forward and backward property, a simple, lightweight, but robust security protocol which achieves data confidentiality, data authentication, data integrity, and data freshness with low overhead and simple operations is presented. Furthermore, an RC4 state based hash function for the generation of message authentication code (MAC) is presented. The proposed protocol and hash function are ideal solution for resource-constrained applications where the communication nodes have limited power resources and computational capabilities, and can be widely used in the applications of one-to-one communications as well as broadcasting and multicasting.**

***Keywords* - RC4; RC4 State; Secure Protocol; Secure Data Transmission; Hash Function; Backward Property of RC4 State; Resource Constrained Devices**

I. INTRODUCTION

Resource-constrained devices, such as wireless sensor networks (WSNs) and radio frequency identification devices (RFIDs), have been applied to a large number of areas. Security is a critical factor of these applications due to the impact on privacy, trust and control [1].

Size and cost constraints on resource-constrained devices result in corresponding constraints on resources such as limited power supplies, low bandwidth, small sized memories, and limited computing. For example, current wireless sensor devices use simple, battery powered 4-bit or 8-bit processors and small amount of memories to perform a few bit-wise logic and simple arithmetic operations. The traditional cryptographic algorithms are too complex and power hungry to be used by resource-constrained devices. In order to provide secure communication to resource-constrained devices, lightweight is the first concern.

A. Perrig et al presented a set of security protocols named SPINS [2] for wireless sensor networks. SPINS has two security building blocks: SNEP provides data confidentiality, two-party data authentication, as well as data freshness, and μTESLA provides broadcast authentication. SPINS is a typical security protocol for wireless sensor networks, especially it avoids asymmetric cryptography to achieve broadcast authentication. SPINS provides data confidentiality by using RC5 block cipher. According to the analysis, stream ciphers are almost always faster and use far less code than the block ciphers [3, 4]. The operations provided by most resource-constrained devices are limited to bit-wise logic operations and simple arithmetic operations which are not power enough for block ciphers.

State Based Key Hop (SBKH) protocol is an RC4 state based secure protocol [5]. SBKH introduced offset to avoid RC4 weak key issue, but SBKH suffers from fake acknowledgement attacks and does not support multicasting and broadcasting due to the strong resynchronization requirement. As a stream cipher, the same key of the SBKH can never be reused. A new key to be generated and distributed for every packet is too difficult to be implemented for resource-constrained communication networks.

The rest of the paper is organized as follows. We first define the RC4 state and introduce its forward and backward property in Section 2. In Section 3 we present an RC4 state based secure protocol which maintains all advantages of the RC4 stream cipher and does not require a new key for every packet. An RC4 state based hash function is presented in Section 4. The analysis of the proposed secure protocol and hash function is provided in Section 5. The section 6 concludes this paper.

II. RC4 STATE AND ITS FORWARD AND BACKWARD PROPERTY

RC4 is a variable key-size stream cipher based on a 256-byte secret internal state and two one-byte indexes. The data are encrypted by XORing data with the key stream which is generated by RC4 from a base key. RC4 consists of two parts which are key-scheduling algorithm (KSA) and pseudo-random generation algorithm (PRGA). For a given base key, KSA generates an initial permutation state. PRGA is a repeated loop procedure and each loop generates a one-byte pseudo-random output as the stream key. That is: at each loop, a one-byte stream key is generated and it is XORed with one-byte of the plaintext, in the meantime a new 256-byte permutation state as well as two one-byte indexes i and j are updated, which defined by (S_{k+1}, i_{k+1}, j_{k+1}) = PRGA(S_k, i_k, j_k) where $i_{k+1}$and j_{k+1} are the indices and S_{k+1} is the state updated from i_k, j_k, and S_k by applying one loop of PRGA.

The operation codes of KSA and PRGA are listed in Figure 1. It has been shown that 256-byte permutation state generated by KSA and PRGA have excellent randomness [3]. In order to strengthen the security and avoid the possible bias in the distribution of the initial base key, the initial permutation state obtained from KSA should run a number of loops of the PRGA (offset) to get a new permutation state before encryption or decryption [3, 5].

On the issue of security strength of RC4, a number of papers have been published to analyze the possible methods of attacking RC4, but none is practical with a reasonable key length, such as 128 bits [4]. WEP is an algorithm to secure IEEE 802.11 wireless network and it requires the use of RC4. Beginning in 2001, several serious weakness (including [6, 7]) were reported and they demonstrate that WEP protocol is vulnerable in a number of areas. In essence, the problem is not in RC4 itself but in the way to generate the key and in how to use the key for RC4 encryption.

In this paper, we present an RC4 state based security protocol which maintain the simplicity and efficiency of the RC4, and eliminate its limitations. It is based on the backward property of RC4 states.

First, we define the reversed algorithm of PRGA (IPRGA) which has the same operations of PRGA but in the reversed order and has no stream key generated. Figure 2 shows the operation codes of IPRGA.

Definition 1: An RC4 state is a set of (S, i, j) where S is a 256-byte permutation of {0, 1, ---, 255} and i and j are one byte indices obtained from algorithm of PRGA and IPRGA.

Theorem 1: If (S^*, i^*, j^*) = $PRGA^k$(S, i, j), then it has (S, i, j) = $IPRGA^k$ (S^*, i^*, j^*) where $PRGA^k$ denotes applying PRGA by k loops (same for $IPRGA^k$).

Theorem 1 indicates that any former RC4 state can be recovered from a later RC4 state by applying IPRGA corresponding loops. The proof of the Theorem 1 is provided in Appendix. Thereby, we can conclude that any RC4 state can be forward to a new RC4 state by PRGA and backward to a previous RC4 state by IPRGA.

```
$KSA(S)$
$Initialization:$
$for\ i = 0\ to\ 255\ do$
    $S[i] = i;$
    $T[i] = K[i \bmod keylen];$
$Initial\ Permutation\ of\ S:$
$j = 0;$
$for\ i = 0\ to\ 255\ do$
    $j = (j + S[i] + T[i]) \bmod 256$
    $S[i] \leftrightarrow S[j]$

$PRGA(S)$
$Generation\ loop:$
    $i = (i+1) \bmod 256$
    $j = (j + S[i]) \bmod 256$
    $S[i] \leftrightarrow S[j]$
    $Output: z = S[(S[i] + S[j]) \bmod 256]$
```

Figure 1: The operation codes of KSA and PRGA

```
$IPRGA\ (S, i, j)$
$Generation\ loop:$
    $S[i] \leftrightarrow S[j]$
    $j \leftarrow (j - S[i] + 256) \bmod 256$
    $i \leftarrow (i-1) \bmod 256$
```

Figure 2: The operation codes of IPRGA

III. RC4 State Based Secure Protocol

In this section, an RC4 state based secure protocol is proposed. Some terminologies and notations used in this section are described first below.

Each communication node keeps an RC4 state called corresponding RC4 state (CRS). The CRS is to be updated after the encryption or decryption of each fixed length data packet.

The sequence counter (SC) stores a number (initially zero) and increases by one each packet. The sender and each receiver have their own sequence counter (SC). For the sender, the SC number is increased by one for each new fixed-length data packet. For the receiver, by checking the SC number of a new incoming fixed-length data packet, operations are performed as follows: if the incoming SC number is one greater than the stored SC number, then the receiver updates its SC number to match the new value and decrypts the data segment by its CRS directly; otherwise, the receiver needs to compute the correct CRS by applying corresponding loops of PRGA or IPRGA to its CRS, and then to decrypt. The receiver has to update its SC number to match the new value which it received.

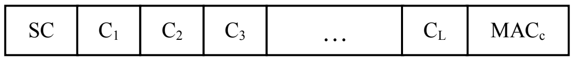

Figure 3: The (encrypted) fixed-length data packet

During the handshaking phase (e.g. before the initial transmission or the base key change), the sender and each receiver share the RC4 base key, the MAC key, the offset value O. Then both sides apply KSA to generate an initial permutation of the RC4 state S_0, and then apply offset (O) loops of PRGA to S_0 to generate a new RC4 state as the CRS for key stream generation. At the same time, both sides set their sequence counters to zero (SC=0).

The proposed protocol requires that the length of the data packets is fixed which includes SC value, L bytes data segment and MAC segment as shown in Figure 3.

Figure 4 depicts the two-step operations which are executed by the sender.

(1) The sender places the next L bytes plain data into data segments and SC into SC segment which is one larger than the previous SC value. The sender then calculates the MAC checksum by inputting SC segment, data segment and MAC key, and then fills the MAC checksum into the MAC segment.

(2) The sender encrypts the data in data segment and MAC segment one byte at a time according to sender's CRS.

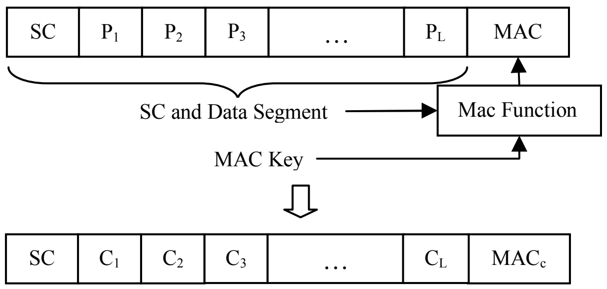

Figure 4: The sender's operations

Figure 5 depicts the two-step operations which are executed by the receiver.

(1) By reading the SC value of the coming fixed-length data packet, the receiver can find the right CRS based on the forward and backward property of RC4 states. If the SC value of the packet received is one larger than the SC value of receiver then the receiver decrypts the data in data segment and MAC segment according to receiver's CRS. Otherwise some calculations are needed in order to get right CRS for decryption. For example, if the coming SC value is the same as the receiver's SC value, then the receiver should apply one loop of IPRGA to its CRS. The receiver then decrypt the data which has been encrypted (it is the data segment and MAC segment here) according to its CRS.

(2) The receiver verifies the MAC checksum. The receiver verifies this fixed-length data packet by re-computing the MAC checksum. If the recomputed MAC checksum is the same as what it was received, the MAC checksum is verified.

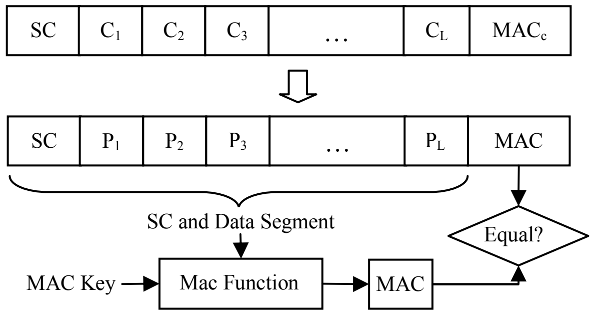

Figure 5: The receiver's operations

In many network-based applications, it cannot guarantee that packets are received in the right order or there is no lost packet. Due to the backward and forward property of RC4 states, the proposed protocol allows that packets receive in an arbitrary order, or resend a lost packet.

Note that, whereas WEP and WPA 1.0 reinitialize the RC4 state for every packet and generate the cipher stream from the initialized RC4 state, the proposed protocol does not reinitialize RC4 states frequently. Instead, the proposed protocol maintains the same RC4 base key for a duration known to each communicating node. This requires the initialization of the RC4 state (by KSA) to be carried out only when the base key changes.

IV. RC4 State Based Hash Function

There are a number of well known secure hash functions including SHA (SHA-512), MD5, Whirlpool and Ripemd-160. Some of them have been widely used and become key components of security protocols. However none of them is applicable to resource-constrained applications. For example, SHA-512 requires each block (1024 bits) to be processed by 80 rounds. Each round needs a large number of 64-bit applications [4].

In this section, we present an RC4 state based hash function which can be used to generate Message Authentication Code (MAC). The proposed hash function significantly reduces the computation overhead and achieves all requirements of a secure hash function. Its security is based on the pseudo-randomness of RC4 states. In the proposed RC4 state based hash function, we use the notations of KSA* and PRGA*, each of which is a part of KSA or PRGA procedures that are listed in Figure 6.

```
$KSA*(S)$
$for\ i = 0\ to\ 255\ do$
$j = (j + S[i] + K[i]) \bmod 256$
$S[i] \leftrightarrow S[j]$
```

RC4 State, Key
KSA*
RC4 State

```
$PRGA*(S)$
$Generation\ loop:$
$i = (i + 1) \bmod 256$
$j = (j + S[i]) \bmod 256$
$S[i] \leftrightarrow S[j]$
```

RC4 State, Offset
PRGA*
RC4 State

Figure 6: The operation codes of KSA* and PRGA* and their graphic representations

The input of KSA* is an RC4 state (e.g. an RC4 state which is generated by PRGA*) and a 64-byte key (K[0], K[1], ---, K[63]); the output is a new RC4 state. The input of PRGA* is an RC4 state and an integer to indicate how many loops the PRGA* applies; the output is another new RC4 state. Essentially KSA* is the same as KSA but it does not initiate the RC4 state at the beginning, and PRGA* is the same as PRGA but it does not generate the key stream.

The input of the proposed hash function is a message with a maximum length of less than 2^{64} bits and the output is a 258-byte message digest. The procedure of the proposed hash function can be described by the following two steps:

Step 1: Divide input message into 512-bit blocks: The input message is divided into 512-bit blocks. The message is padded so that its length is N times of 512 bits. The padding bit is 0 and the number of padding bits is in the range of 0 to 512.

Step 2: Process message blocks: The first 512-bit message block m_1 is processed by KSA and PRGA* in sequence. The input of the KSA is m_1 as the secure key and the output is an RC4 state named $State_{m1}$. The input of the PRGA* is $State_{m1}$ (as the initial state for PRGA*) and the output is an RC4 state named $State_1$ which is generated by applying (m_1 mod 2^5) loops of PRGA*. The output RC4 state ($State_x$) will be the initial RC4 state of KSA* to process next message block. The second 512-bit message block is processed by KSA* and PRGA* in sequence. The input of KSA* are the $State_2$ as the initial RC4 state and m_2 as the secure key and the output is $State_{m2}$. As the previous operation, $State_{m2}$ will be used as the input for the corresponding (m_2 mod 2^5) loops of PRGA* operation to generate $State_2$. The rest of blocks do the same process as the second block with corresponding input. $State_N$ is the 258-byte output of this RC4 state based hash function. The process of this step is depicted in Figure 7.

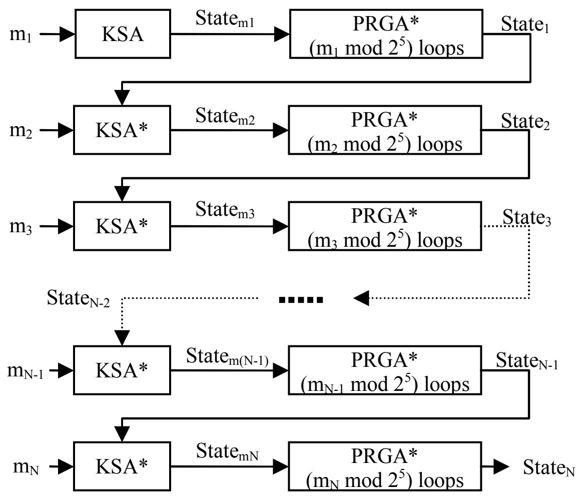

Figure 7: The procedure of the proposed RC4 state based hash function

Note that the 258-byte output of the proposed hash function can be used as multiple new session keys. In case less number of bits of hash value is needed, any method to extract the number bits from 258 bytes of the hash function output can be used.

V. Analysis and Implementation

The proposed secure protocol is based on the RC4 algorithm with a number of improvements. The improvements include using offset and the forward and backward property of RC4 states.

The forward and backward property of RC4 states is an intrinsic property of the RC4 algorithm. Any previous RC4 state can be recovered from the current or later RC4 state by simple addition and subtraction operations, as well as by swap operations. Since the proposed protocol simply uses this property and makes no changes to the original RC4, using the forward and backward property does not reduce the security strength of RC4.

Offset is used in the proposed protocol to discard the first few bytes of the key stream, and data encryption or decryption begins at some later position. The purpose of using offset is to avoid the weaknesses of the initial RC4 states [3]. While the inclusion of offset is not a fundamental change to RC4, the security performance is improved. Offset does not reduce the security strength of RC4.

For proposed RC4 state based hash function, we have implemented a simulation to analyze the randomness of the RC4 state generation for KSA* and PRGA*. Simulation results show that the generation of a new RC4 state by KSA* or PRGA* maintains the same randomness as KSA or PRGA does.

Based on the analysis above, we believe that the proposed protocol uses RC4 in a way which makes effective use of RC4's strengths and reduces its weaknesses.

In the respect of the performance, the proposed protocol runs simply and efficiently because of the benefits of RC4 and the new features. As we indicate in Section 2, RC4 is simple and fast and the overhead of using RC4 is remarkably low. All operations in the proposed protocol are simple swap, XOR, and addition and subtraction operations. Furthermore, the proposed protocol uses RC4 more efficiently than the regular way. By using the forward and backward property of RC4 states, the proposed protocol does not require key initialization frequently. This translates directly to some simplification and power saving, as well as improved performance over other original RC4 based protocols. For example, if the length of the data segment of the packet is 256 bytes, since KSA is unnecessary to be applied for every new packet, the encryption/decryption process of the proposed secure protocol requires about half time than the time required by the original RC4 based protocol, such as WEP.

Table 1 and Table 2 indicate the comparisons of the execution times among AES-based protocol, RC5-based protocol, WEP and the proposed protocol. The comparisons are on an 8-bit CPU. For Table 1, t_1 is time for a logic operation or a simple arithmetic operation, t_2 is time for one time memory access. For Table 2, we assume $t = t_2 = 2t_1$.

Table 1: The Comparisons of the Execution Times on an 8-bit CPU (t_1 is time for a logic operation or a simple arithmetic operation, and t_2 is time for one time memory access)

Data Length	64 bytes	128 bytes	256 bytes	512 bytes	1024 bytes
AES based	$5824t_1+6272t_2$	$11648t_1+12544t_2$	$23296t_1+25088t_2$	$46592t_1+50176t_2$	$93184t_1+100352t_2$
WEP	$1024t_1+2816t_2$	$1280t_1+3328t_2$	$1792t_1+4352t_2$	$2816t_1+6400t_2$	$4864t_1+10496t_2$
RC5 based	$14784t_1+832t_2$	$29568t_1+1664t_2$	$59136t_1+3328t_2$	$118272t_1+6656t_2$	$236544t_1+13312t_2$
Proposed	$256t_1+512t_2$	$512t_1+1024t_2$	$1024t_1+2048t_2$	$2048t_1+4096t_2$	$4096t_1+8192t_2$

Table 2: The Comparisons of the Execution Times on an 8-bit CPU (assuming $t = t_2 = 2t_1$)

Data Length	64 bytes	128 bytes	256 bytes	512 bytes	1024 bytes
AES-based	9184t	18368t	36736t	73472t	146944t
WEP	3328t	3968t	5248t	7808t	12928t
RC5-based	8224t	16448t	32896t	65792t	131584t
Proposed	640t	1280t	2560t	5120t	10240t

The proposed RC4 state based secure protocol has been implemented. It consists of two major parts. One is on sender side, and another is on receiver side. Our simulation shows that the proposed protocols work well to deal with the delayed or lost packet. Compared with the ordinary RC4 based secure protocol, the proposed one are almost twice faster than the original one. Our simulation also shows that the proposed protocol is secure and has the ability to protect against a number of possible attacks including acknowledge attack, replay attack, and modified packet attack. We have applied the proposed protocol to a multicasting environment where the numbers are dynamically changed. The experiment works correctly and smoothly.

VI. Conclusion

Resource-constrained applications, such as WSNs and RFIDs, are being deployed widely and become more and more important. Security is a critical factor of these applications so a simple, efficient, and robust secure protocol for communications on those devices is in widely demand. This paper focuses on the designs of the secure protocol and hash function for secure data transmission on resource-constrained devices which is based on the RC4 state and its forward and backward property.

Due to its simplicity, efficiency and high security, the proposed protocol and hash function are comparable with well-known security protocols and hash function. They are very simple to implement and efficient in both hardware and software, and is compatible with different levels of security. It can be widely used in many network applications. Furthermore, the proposed RC4 state based secure protocol and hash function are relatively independent parts and can be served as one of the key components for other secure applications.

REFERENCES

[1] J. P. Kaps, G. Gaubatz, and B. Sunar. Cryptography on a Speck of Dust. *IEEE Computer Magazine*, Feb. 2007, pp38-44.

[2] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security Protocols for Sensor Networks. *Wireless Networks*, 2002, pp521-534.

[3] I. Mantin. Analysis of the Stream Cipher RC4. Master's thesis, The Weizmann Institute of Science, 2001.

[4] W. Stallings. *Cryptography and Network Security*, Principles and Practice. Prentice Hall, 2003.

[5] S. Mitchell and K. Srinivasan. State Based Key Hop Protocol: A Lightweight Security Protocol for Wireless Networks. *Proc. 1st ACM international workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks*, 2004, pp112-118.

[6] A. Stubblefield, J. loannidis, and A.D. Rubin. Using the Fluhrer, Mantin, and Shamir Attack to Break WEP. AT&T Labs Technical Report, 2001.

[7] S. Fluhrer, I. Mantin, and A. Shamir. Weakness in the Key Scheduling Algorithm of RC4. *Proc. Workshop in Selected Areas of Cryptography, 01.*

APPENDIX

Theorem 1: For any r, $n \geq r \geq 0$, we have:

$$IPRGA^r (S_n, i_n, j_n) = (S_{n-r}, i_{n-r}, j_{n-r})$$

Proof.

The proof is conducted by induction on integer r

Base Case:

It is true when r=0: $IPRGA^0 (S_n,i_n,j_n)=(S_n,i_n,j_n)$

Inductive Step:

Assume that for any $r \leq k$ we have:

$IPRGA^k (S_n, i_n, j_n) = (S_{n-k}, i_{n-k}, j_{n-k})$

Consider case of r = k+1

$IPRGA^{k+1} (S_n, i_n, j_n) = IPRGA (IPRGA^k (S_n, i_n, j_n))$

$= IPRGA (S_{n-k}, i_{n-k}, j_{n-k})$

Let $IPRGA (S_{n-k}, i_{n-k}, j_{n-k}) = (S^*, i^*, j^*)$

According to PRGA and our notation, we have

$PRGA (S_{n-k-1}, i_{n-k-1}, j_{n-k-1}) = (S_{n-k}, i_{n-k}, j_{n-k})$

where $i_{n-k} = i_{n-k-1}+1$ and $j_{n-k} = j_{n-k-1} + S_{n-k-1}[i_{n-k-1}+1]$

$S_{n-k} [m] = S_{n-k-1}[m]$

where $m \neq i_{n-k-1}$ and $m \neq j_{n-k-1} + S_{n-k-1} [i_{n-k-1} +1]$

$S_{n-k} [i_{n-k-1} + 1] = S_{n-k-1} [j_{n-k-1} + S_{n-k-1} [i_{n-k-1} + 1]]$

$S_{n-k} [j_{n-k-1} + S_{n-k-1} [i_{n-k-1} +1]] = S_{n-k-1} [i_{n-k-1} + 1]$

According to $IPRGA (S_{n-k}, i_{n-k}, j_{n-k}) = (S^*, i^*, j^*)$,

we have $S^*[m] = S_{n-k} [m] = S_{n-k-1}[m]$

where $m \neq i_{n-k-1}$ and $m \neq j_{n-k-1} + S_{n-k-1} [i_{n-k-1} +1]$

and $S^* [i_{n-k-1}+1]= S_{n-k} [j_{n-k-1}+ S_{n-k-1}[i_{n-k-1} +1]]$

$= S_{n-k-1} [i_{n-k-1} + 1]$

$S^* [j_{n-k-1} + S_{n-k-1} [i_{n-k-1} +1]] = S_{n-k} [i_{n-k-1} +1]$

$= S_{n-k-1} [j_{n-k-1} + S_{n-k-1} [i_{n-k-1} +1]$

Thus $S^*=S_{n-k-1}$

$j^*=j_{n-k}–S^*[i_{n-k}]=j_{n-k-1}+S_{n-k-1}[i_{n-k-1}+1]–S_{n-k-1}[i_{n-k-1}+1]=j_{n-k-1}$

$i^*=i_{n-k}-1=i_{n-k-1}$

Therefore, we have $(S^*, i^*, j^*) = (S_{n-k-1}, i_{n-k-1}, j_{n-k-1})$

That is, for any $n \geq r \geq 0$,

we have: $IPRGA^r (S_n, i_n, j_n) = (S_{n-r}, i_{n-r}, j_{n-r})$ □