Cryptanalysis of the Light-Weight Cipher A2U2 First Draft version

Mohamed Ahmed Abdelraheem, Julia Borghoff, Erik Zenner

Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark
{M.A.Abdelraheem,J.Borghoff,E.Zenner}@mat.dtu.dk

1 Introduction

At IEEE RFID 2011, David et al. proposed a new cryptographic primitive for use with RFID [2]. The design is a stream cipher called A2U2. Shortly afterwards, an attack was published on IACR Eprint by Chai et al. [1], claiming to break the cipher in a chosen-plaintext attack using extremely little computational resources. Regrettably, this attack is wrong since it works with an erroneous description of the cipher. In this paper, we show why the attack is wrong and how it can be repaired (Section 4).

Furthermore, in Section 5 we describe a guess-and-determine attack which applies in a known plaintext scenario.

A special design feature of A2U2 is that the number of initialization rounds varies and depends on an internal counter. The number of rounds varies from 9 to 126. In Section 6 we proposed a differential style attack which first enables us to find the counter value determining the number of initialization rounds. Moreover we present an attack that recovers the masterkey in the case that only 9 initialization rounds are used.

2 Description of the Cipher

The cipher's inner state consists of a counter LFSR C (7 bit), two non-linear feedback shift registers A and B (NFSRs, 17 and 9 bit), and a key register K (56 bit). Thus, the total inner state size is 89 bit. Note that we use a different notation than in the original paper, our being more suitable for cryptanalysis.

Updating the counter: In the following, we denote the state of the counter LFSR at time t by $C^t = (C_t, C_{t-1}, \ldots, C_{t-6})$, for $t = 0, 1, \ldots$. The starting state after initialisation is $C^0 = (1, 1, \ldots, 1)$, see below under "Initialisation". The LFSR then uses the feedback recurrence

$$C_t = C_{t-7} + C_{t-4}$$

for updating the state for $t \geq 1$. It is a standard LFSR with maximal period (i.e., $2^7 - 1$).

Updating the NFSRs We denote the state of the NFSRs by $A^t = (A_t, \ldots, A_{t-16})$ and by $B^t = (B_t, \ldots, B_{t-8})$. The update uses an auxiliary variable h_t (defined below under "Updating the key register") and the following non-linear feedback recurrences:

$$B_t = A_{t-17} + A_{t-15}A_{t-14} + A_{12} + A_{t-10}C_{t-7} + A_{t-7}A_{t-6}A_{t-5} + A_{t-4}A_{t-2}$$
$$A_t = B_{t-9} + B_{t-8}B_{t-7} + B_{t-6} + B_{t-3} + h_t + 1,$$

again for $t \geq 1$.

Updating the key register The key register is a rotation register, i.e. the state in time t is a rotated version of the initial state. If we denote the key bits by $(k_0, \ldots, k_{55})$, then each state of the key register is defined as

$$K^t = (k_{5t}, k_{5t+1}, \ldots, k_{5t+55}),$$

where all indices are computed modulo 56.

For each round, the first five bits of the register are stored in a buffer $S^t = (S_0^t, \ldots, S_4^t) = (k_{5t}, \ldots, k_{5t+4})$, and they are used to compute the auxiliary variable h_t as follows:

$$h_t = \mathrm{MUX}_{C_{t-5}}(S_0^t, S_1^t) \cdot \mathrm{MUX}_{C_{t-1}}(S_4^t, A_{t-2}) + \mathrm{MUX}_{C_{t-3}}(S_2^t, S_3^t) + 1,$$

where $\mathrm{MUX}_z(x, y)$ is the multiplexer function that selects x if $z = 0$ and y otherwise.

Initialisation: The A2U2 cipher has a 61-bit key split into two parts: A 5-bit "counter key" and a 56-bit "register key". In addition, the cipher receives a 32-bit random number. Key and random number are written into the registers as follows:

- **Counter register:** The 5 least significant key and random bits are xored. The resulting 5-bit vector is written into the 5 least significant bits of the counter LFSR (using the above notation). The second MSB is set to 1, the MSB is set to 0.
- **NFSRs:** The next 26 key and random bits are xored and written into the NFSR cells.
- **Key register:** The 56 register key bits (to some extend the same that were used for NFSR initialisation) are stored in the key register.

Now the cipher is clocked until the counter register reaches the all-one state. This happens after 9-126 clockings. The resulting state is called the initial state.

Output generation: The cipher deploys a form for irregular output mechanism; it outputs either encrypted plaintext bits or pseudo-random bits depending on the content of NFSR cell A_t. Plaintext bits have to "wait" until $A_t = 1$ before being encrypted. If we denote the plaintext string by $P = (P_0, P_1, \ldots)$ and if we define $\sigma(t) = \sum_{i=0}^{t-1} A_t$ with $\sigma(0) = 0$, then the output of the cipher in round t is:

$$Y_t = \mathrm{MUX}_{A_t}(B_t + C_t, B_t + P_{\sigma(t)}).$$

3 Cryptanalysis

3.1 Useful Properties

In the following, we point out some properties of the cipher that will be used by our attacks.

Known counter: Since the counter has the all-one state after initialisation, the attacker can compute all successive counter states and does know the bit C_t for any $t \geq 0$. This simplifies significantly some of the algebraic equations described above.

Chosen randomisation vector: Instead of using a nonce that is chosen solely by the encrypting party as is common practice with stream ciphers, the random vector used for initialisation is generated in collaboration between sender and receiver. According to [2], both sender and receiver each choose a 32-bit random number which is then sent over the communication channel. Both inputs are then combined by xor into the actual randomisation vector. Note that this procedure enables an active attacker to choose the randomisation vector: He waits for the legitimate party's input and chooses his own such that the xor sum will be the desired number. In particular, he can introduce arbitrary differences between randomisation vectors and even force the encryption device to use the same randomisation vector twice, thus violating an important design principle for stream ciphers.

Chosen plaintext: Due to the unusual output generation, a chosen plaintext attack is more powerful than a known plaintext attack. This is another difference to traditional stream cipher designs, where no extra information is gained if the attacker is allowed to choose the plaintext himself.

4 A Chosen Plaintext Attack

4.1 Disproving the Chai/Fan/Gong Attack

In [1], a very efficient chosen plaintext attack against A2U2 is proposed. However, as we are going to show in this section, the attack contains a flaw that makes it unapplicable against the real A2U2 cipher.

Attack Idea: If the attacker could freely choose one plaintext bit for each clock, then he can write the output equation as follows:

$$\begin{aligned} c_t &= \mathrm{MUX}_{A_t}(B_t + C_t, B_t + p_t) \\ &= \mathrm{MUX}_{A_t}(C_t, p_t) + B_t. \end{aligned}$$

Depending on the amount of knowledge the attacker has about the plaintext, he can now learn more about the inner state.

If the attacker can choose the plaintext, he can start by encrypting a plaintext that is identical to the counter sequence. In this case, the above equation simplifies to

$$\begin{aligned} c_t &= \mathrm{MUX}_{A_t}(C_t, p) + B_t \\ &= p + B_t, \end{aligned}$$

meaning that the attacker can learn the whole sequence $(B_t)_{t\geq 0}$.

Next, he encrypts a plaintext that is the bitwise inverse of the counter sequence. This allows him to distinguish for every ciphertext bit whether C_t or p was encrypted, providing the attacker with the full sequence $(A_t)_{t\geq 0}$.

Now he has the sequences produced by the LFSR and by both NFSRs. All that remains is to test for each round which key bit gives the correct NFSR update. This can be done in unit time, yielding an extremely efficient attack.

The Catch: However, the initial assumption of the above attack is wrong, invalidating the whole cryptanalysis. The problem is that plaintext is not used at a rate of 1 bit per round. It is not possible to choose a plaintext bit for each round, because (1) some plaintext bits are used in several rounds and (2) without knowledge of the sequence $(A_t)_{t\geq 0}$, it is impossible to say in which rounds a plaintext bit will be used.

4.2 A Leak in the Output Function

However, as it turns out, this attack can be repaired. In the following, we will demonstrate a leak in the output function that can even be used for general known-plaintext attacks and will then expand this weakness into a chosen plaintext attack that reminds of the one described above but is actually functional.

Known plaintext: Assume that the inner sequences A_t and B_t are statistically close to random. Then in particular, $\Pr(A_t = 0) = 1/2$ for all t. We can now consider two cases for the output function:

- If $A_t = 0$, then $Y_t = B_t + C_t$. Since we know C_t, we can rewrite this as $B_t = Y_t + C_t$. For $A_t = 0$, this is always true.
- If $A_t = 1$, we have $Y_t = B_t + P_{\sigma(t)}$, with $P_{\sigma(t)}$ unknown. If we assume that $P_{\sigma(t)} = C_t$ with probability $1/2$, then the equation $B_t = Y_t + C_t$ is also true with probability $1/2$.

In total, the equation $B_t = Y_t + C_t$ is thus met with probability $1/2 + (1/2)^2 = 3/4$, i.e. by observing the keystream and knowing the behaviour of the counter LFSR, we can predict the inner stream $(B_0, B_1 \ldots)$ with probability $3/4$ per bit.

Chosen plaintext: Note that when we can choose the plaintext, we can increase the probability of $P_{\sigma(t)} = C_t$ and thus the probability of the equation $B_t = Y_t + C_t$ being correct.

As an example, consider the first 5 output bits of the LFSR, which are $(1, 0, 0, 0, 0)$. Thus, if we choose a plaintext $(1, 0, 0, 0, 0)$, then $P_{\sigma(t)} = C_t$ is true with probability 1 for the first bit, $1 - 1/2$ for the second, $1 - 1/4$ for the third, $1 - 1/8$ for the fourth, and $1 - 1/16$ for the fifth bit. Thus, we can predict the inner state bits $(B_0, \ldots, B_4)$ with probabilities $(1, 3/4, 7/8, 15/16, 31/32)$.

The most useful plaintexts for this kind of analysis seem to be $(0, 0, ...)$ and $(1, 1, ...)$, since for them, the attacker knows exactly the bit $P_{\sigma(t)}$ for every time slot t. Let us start with the all-zero sequence. The attacker knows that the plaintext sequence $(P_{\sigma(t)})_{t\geq 0}$ consists only of zeros. He now looks at all time slots t with $C_t = 0$. For those time slots, it holds that $B_t = Y_t$, independent of the choice of A_t. Thus, he learns about half of the bits of the sequence B.

The remaining bits can be learned using the all-one sequence. In this case, in all positions where $C_t = 1$, the attacker learns $B_t = Y_t + 1$, also independent of A_t. Thus, he has fully reconstructed the sequence B.

What is more, he can also use this new information to learn the sequence A as well. For every time slot, he either picks the ciphertext bit Y_t corresponding to the plaintext bit $P_{\sigma(t)} \neq C_t$. If it holds that $B_t = Y_t + C_t$, then $A_t = 0$, otherwise $A_t = 1$.

After this step, the attacker knows the sequences generated by all three registers A, B, and C. The remaining attack proceeds as follows. Knowing the sequences A, B and C the attacker can determine the values h_t because

$$h_t = B_{t-9} + B_{t-8}B_{t-7} + B_{t-6} + B_{t-3} + A_t + 1.$$

Furthermore, it holds that

$$h_t = \mathrm{MUX}_{C_{t-5}}(S_0^t, S_1^t) \cdot \mathrm{MUX}_{C_{t-1}}(S_4^t, A_{t-2}) + \mathrm{MUX}_{C_{t-3}}(S_2^t, S_3^t) + 1,$$

where C_{t-i} for $i = 1, 3, 5$ and A_{t-2} are known. This equation is at most quadratic and in about half of the cases (when $C_{t-1} = 1$) it is linear. Determining 56 values of h_t yields a fully determined quadratic Boolean equation system, which can be solved by e.g., using Gröbner basis techniques. As about half of the equation are linear, a linear equation system can be obtained after determining 112 values of h_t. After 11 clockings of the algorithm the key register is rotated once and the key bits are reused, thus it can happen that the same equation is generated twice. However, experiments showed that this does not happen frequently, thus we expect that observing around 120 values of h_t is sufficient to generate a fully determined linear equation systems in 56 unknowns.

Effort: The attack requires two chosen plaintext of length around 120 bits which are encrypted using the same key and the random number in the initialization in order to recover secret key bits (excluding the 5 bits which are used to initialize the counter). The main computational effort consists in solving a linear Boolean equation system which can be done in well under a 1 second. Thus, in the chosen plaintext scenario, the cipher must be considered as completely broken.

5 Guess-and-Determine attack

In this section a we discuss known plaintext attack which is in general a more likely scenario than a chosen plaintext attack. When we know but are not allowed to choose the plaintext we cannot use the same trick as in Section 4 to determine the sequence $(B_t)_{t\geq 0}$. We cannot simply calculate this sequence for a given plaintext/ciphertext pair because we do not know which bits of the ciphertext correspond to the plaintext bits. This is controlled by register A. The idea of this attack is to guess the sequence of $(A_t)_{t\geq 0}$, meaning we guess at which positions of the ciphertext a plaintext bit was used. These guesses are used to determine additional bits of register B and then later on the value of h_t. As we know the value of the counter at any time during the encryption process, given h_t and A_{t-2} we obtain a Boolean equation in the key bits which is at most quadratic and contains at most three variables. If we are able to collect sufficiently many of those equations we will be able to recover the key bits by solving the equation system.

We denote by A_0 the content of the last cell of the first NFSR at the time when the ciphertext generation starts.The attack is divided into three parts of guessing bits.

In the first part we guess the value A_t for $t = 0, \ldots, 8$ for 8 consecutive clockings of the algorithms. Depending on our guess we know if the counter bit or a plaintext bit was used to generate the corresponding ciphertext bit and we can determine the value of B_t for $t = 0, \ldots, 8$. After guessing 9 bits we know the full second NFSR and about the lower half of the first NFSR.

In the next part we continue guessing the value of A_t for $t = 9, \ldots, 16$ and determine the value of B_t for $t = 9, \ldots, 16$. Additionally, we obtain the value of h_t for $t = 9, \ldots, 16$ and the corresponding Boolean equation in the key bits because it holds that

$$h_t := A_t + B_{t-9} + B_{t-8}B_{t-7} + B_{t-6} + B_{t-3} + 1, \tag{1}$$

the full register B is known and we guessed the value of A_t. After the second part of the attack both registers are known and we have already obtained 8 equations.

In the third part we want to determine the value of h_t for further clockings of the cipher. The full register A is known and in order to update register B only bits of register A are used. This means we can update register B and know the value which was use to encrypt next ciphertext bit (bit 17, 18 etc). Furthermore, we know the counter value C_t and the plaintext bit p that might have been used (according to our guess). As mentioned before we want to determine the value of h_t and obtain the corresponding equation. Using equation (1) we need to determine A_t in order to obtain h_t. This can be done in two ways. Either we have to guess A_t as we did before, or we can simply calculate the value of A_t. This can be done when $C_t \neq P_{\sigma(t)}$ because it holds that

$$Y_t + B_t + C_t = A_t(C_t + P_{\sigma(t)}).$$

Effort: We need to collect at least 56 equations to determine a unique solution in 56 key variables. In the first two parts of the attack we guess 17 bits and obtain 8 equations. We expect that we have to guess every second value for A_t in the third part of the attack. Thus, we expect that we have to guess 24 further bits in the third part of the attack in order to obtain a fully determined equation system. This leads to a complexity of 2^{41}.

This equation system is non-linear and therefore it might not have a unique solution even though is is fully determined. However, it is often sufficient to add a few extra equations to get a unique solution. This will slightly raise the complexity of the attack.

However, in order to give an more precise estimate of the complexity an implementation is necessary (work in progress) because of the fact that the key register is rotate once after 11 clockings of the algorithm and thus key bits are reused. This property will on the one hand increase the complexity of the attack for the correct guess but on the other hand enable us to discard wrong guesses in an early stage. After producing 11 ciphertext bits the key register has been rotated once, that means the key bits will be reused when generating more equations. This leads to rounds where we guess or determine the value of A_t but do not get a new equation, thus do not gain extra information about the key. This is especially true for the correct guess and means that it is necessary to guess extra bit in order to obtain a fully determined system. For a wrong guess however this might be to our advantage because it is very likely that when the same polynomial is generated the RHS differs. Thus, we get contradicting equations and can abort the guess.

In general, for a wrong guess the equation system will not have a solution. The inconsistency might be very obvious as mentioned above, but it might also be necessary to solve a non-linear Boolean equation system. Therefore, we have to make a trade-off how often we want to check if the system is still solvable.

Experiments to estimate the actual attack complexity are currently carried out and the paper will be updated as soon as we ran a sufficient number of tests.

6 Targeting the low number of initialization rounds

6.1 Recovering the last five key bits

A2U2 has a secret number of initialization rounds. There are 32 possible choices for the number of initialization rounds that varies from 9 to 126 where each choice is specified by the 5 LSBs of the tag's random number and the reader's random number. The attack requires for each of the possible 32 initialization 2^9 state pairs with a good difference (sparse characteristic). Under these states we encrypt a single bit of plaintext which is equal to the first bit of the counter, C_0, at the time when the encryption starts. Then we can distinguish the 2^9 state pairs corresponding to 9 rounds of initialization by observing a bias in the difference of the first bit of the corresponding 2^9 ciphertext pairs, $\Delta(Y_0)$, which is equal to the difference in $\Delta(B_0)$.

Experiments show that $\Pr(\Delta(B_0) = 0) > 0.7$ for 9 initialization rounds when the differences $\Delta(A) = 10000000000000000$ in A and $\Delta(B) = 100000100$ in B are used. The bias is smaller when more initialization rounds are applied. We observe the strong bias in B_0 for 9 rounds because in only 9 rounds the difference cannot propagate through state and does not spread out sufficiently. After having distinguished the 2^9 state pairs corresponding to 9 rounds of initialization, we can consequently find the five last secret key bits.

6.2 Recovering subkey bits and master key bits

The following attack targets plaintext/ciphertext generated using 9 rounds of initialization and exploits the key scheduling used to generate the subkey bits, h_t. The attacker starts by guessing the 26 master key bits used in initializing registers A and B and then at each round he guesses one subkey bits if $C_{t-1} = 0$, or two master key bits if $C_{t-1} = 1$ (since A_{t-2} will then be used to generate h_t) or no bits when all the key bits involved in the generation of the round subkey bit are from the 26 master key bits used in initializing registers A and B. The cipher is initialized using 9 rounds and then a 5-bit plaintext is encrypted, so in total the cipher runs for 14 rounds. Without loss of generality we assume that the starting key position is 2, so the key bits at positions 2 to 27 are used in initializing registers A and B. Table 1 shows the key bits positions used from round 0 to round 13, the value of C_{t-1} and the number of guessed subkey/key bits. The table also shows that we have to guess 12 subkey/key bits plus the 26 master key bits used in initializing the registers. So in total we have a time complexity equal to 2^{38} to recover 32 master key bits and 6 subkey bits and when using plaintexts of length 5-bit we need only $\lfloor \frac{38}{5} \rfloor = 7$ plaintext/ciphertext pairs to find the right key guess. The remaining master key bits can be recover using a brute force search. In order for the above attack to work we need to find only 7 plaintext/ciphertext pairs whose initial state pairs are initialized with position 2 as a starting key position and are generated using 9 initialization rounds which we can easily find using the last five secret key bits recovered in the previous attack.

r	S_0	S_1	S_2	S_3	S_4	C_{t-1}	G	r	S_0	S_1	S_2	S_3	S_4	C_{t-1}	G
0	28	29	30	31	32	0	1	7	7	8	9	10	11	1	0
1	33	34	35	36	37	0	1	8	12	13	14	15	16	1	0
2	38	39	40	41	42	0	1	9	17	18	19	20	21	1	0
3	43	44	45	46	47	1	2	10	22	23	24	25	26	0	0
4	48	49	50	51	52	1	2	11	27	28	29	30	31	0	1
5	53	54	55	0	1	1	2	12	32	33	34	35	36	0	1
6	2	3	4	5	6	1	0	13	37	38	39	40	41	0	1

Table 1. List of the masterkey bits used in the subkey generation of each round, together with the counter value and the number of require guesses. $r \equiv$ round no, $G \equiv$ number of subkey/key bits that are guessed.

References

1. Q. Chai, X. Fan, and G. Gong. An ultra-efficient key recovery attack on the lightweight stream cipher A2U2. http://eprint.iacr.org/2011/247, 2011. Version published: 20110518:133751 (posted 18-May-2011 13:37:51 UTC).
2. M. David, D.C. Ranasinghe, and T. Larsen. A2U2: A stream cipher for printed electronics RFID tags. In *Proceedings of IEEE RFID 2011*, pages 240–247, 2011. to appear.