# HIT: A Hidden Instruction Trojan Model for Processors

Jiaqi Zhang[1,2] Ying Zhang[1*] Huawei Li[2*] Jianhui Jiang[1]
[1] *School of Software Engineering, Tongji University, China*
[2] *SKLCA, Institute of Computing Technology, Chinese Academy of Sciences, China*
zhangjiaqi121@126.com, yingzhang@tongji.edu.cn, lihuawei@ict.ac.cn, jhjiang@tongji.edu.cn

*Abstract*— **This paper explores an intrusion mechanism to microprocessors using illegal instructions, namely hidden instruction Trojan (HIT). It uses a low-probability sequence consisting of normal instructions as a boot sequence, followed by an illegal instruction to trigger the Trojan. The payload is a hidden interrupt to force the program counter to a specific address. Hence the program at the address has the super privileges. Meanwhile, we use integer programming to minimize the trigger probability of HIT within a given area overhead. The experimental results demonstrate that HIT has an extremely low trigger probability and can survive from the detection of the existing test methods.**

## I. INTRODUCTION

Hardware security has always been a key factor in the overall security of computer systems. The hardware chip works at the bottom of a computer system and has the highest operational priority of the entire system. Hence, a hardware Trojan can bypass software security facilities. At the same time, the operations inside the chip are usually transparent to the upper operating system and the network, so a hardware Trojan can easily hide its intrusion behavior. Therefore, hardware security issues are receiving increasing attention from academia and industry.

The instructions between computer hardware and the upper system emerge as a critical access that could be improperly utilized by the hardware Trojan to affect the upper operating system or user applications. The insider hacker or even the design company can take advantage of the reserved opcodes in the instruction set and implement some new features into the processor. The new instructions corresponding to these reserved opcodes are often beyond the design specification, so we call these instructions as illegal instructions, and consider the instruction functions as instruction Trojans (or backdoors). As the instruction Torjan can directly affect the upper system, exploring its possible intrusion behaviors emerges as an important task that will lay the foundation for subsequent researchers to develop test and protection methods.

Many researchers have modeled hardware Trojans and developed test methods [1-7], but modeling intrusion mechanisms of instructions remains a weak point from the known literature. In this paper, we explore an intrusion mechanism using illegal instructions to bypass the security facilities of the operating system and seize the control authority, namely hidden instruction Trojan (HIT). The key contributions are as follows:

- The Trojan mechanism of the proposed HIT is developed using a low-probability sequence consisting of normal instructions as a boot sequence, and the HIT is triggered by an illegal instruction following the boot sequence;
- The Trojan payload is designed as a hidden interrupt, which forces the program counter to a specific address and sets the program at the address to super privilege;
- Integer programming is used to minimize the trigger probability of HIT within a given area overhead.

The rest of the paper is organized as follows. In Section II, the HIT mechanism is described in detail. Section III introduces the methods for optimizing the instruction Trojan. Experimental results are given in Section IV. In Section V, we conclude the paper.

## II. HIDDEN INSTRUCTION TROJAN MODEL

### A. The trigger of HIT

In this work, we model a critical type of hidden instruction Trojans (HIT) that bring in new interrupts to processors. These interrupts are non-maskable privileged interrupts whose ID and handler address are stored inside the hardware (i.e., interrupt processing module). The handler address is directed to the user's address space instead of the kernel to facilitate the hacker invading the computer as a normal user. The trigger of such a HIT requires a boot sequence and a corresponding illegal instruction. The boot sequence is made up by a sequence of normal instructions whose occurrence probability is very low, and can be identified by a finite state machine (FSM). Fig.1 presents the triggering process of the HIT. The FSM always stays at the initial state. When the input instruction is consistent with the boot sequence, the FSM state will migrate from state 1 to state *N*. Once the current input instruction is inconsistent with the instruction in the boot sequence, the FSM state will return to the initial state. If and only if the FSM state migrates to state *N* and then the illegal instruction arrives, the HIT is triggered. Since the Trojan requires a low-probability sequence of normal instructions as its boot sequence, it can be hidden in normal operations without being discovered by the simulation-based test for a long time.

For example, we can implant the hidden instruction Trojan into the decoding pipeline of a miniMIPS processor. The FSM of the Trojan only reads in valid instructions. When a jump (or branch) instruction occurs, the FSM ignores the inserted invalid instructions (i.e., null), but reads in the valid next instruction into the boot sequence. In this way, the boot sequence is immune to the effects of control dependency caused by jump instructions. Fig. 2 presents the triggering process of the instruction Trojan. First, the *Inst* module of the FSM recognizes the low-probability boot sequence, which makes the state on the *State* module synchronously transformed, so that the trigger generates a valid

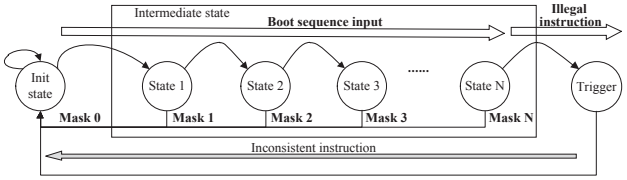



Fig. 1. The instrusion mechanism of illegal instructions.

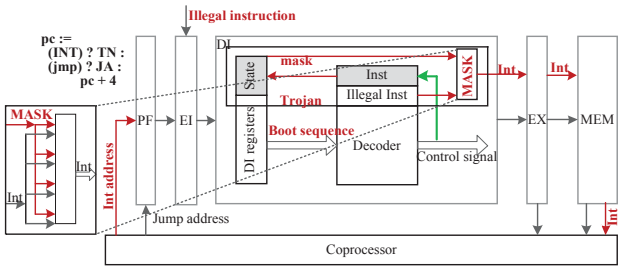



Fig. 2. The miniMIPS processor with a hidden instruction Trojan.

* To whom correspondence should be addressed.

This paper is supported by National Natural Science Foundation of China (NSFC) under grant No. (61432017, 61772199, 61974105, 61532017).

mask for the illegal instruction. We use a multiple-bit mask instead of a single-bit enable signal to weaken the power characteristic of the Trojan. Hence the Trojan can survive from the detection of the side-channel signal analysis. Second, the processor reads in the illegal instruction, and the *Illegal_Inst* module in the decoder recognizes that instruction. Third, the illegal instruction passes the the *MASK* module in Fig. 2 and activates a new interrupt. Therefore, the low-probability boot sequence along with the illegal instruction triggers a hidden privileged interrupt in the processor.

### B. The payload of HIT

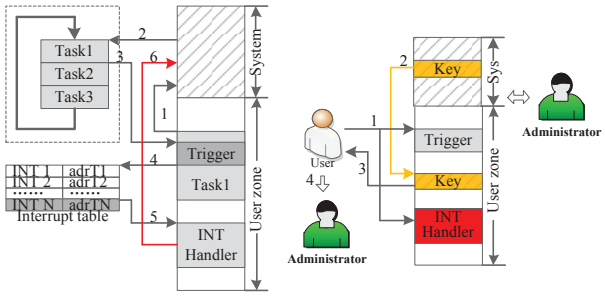
The instruction Trojan can trigger hidden privileged interrupts in the user zone, hence it will seriously jeopardize the rights management of an operating system (OS). An OS divides the address space into system zones and user zones, and its protection mechanism forces the user task to run only in its user zone. However, the protection mechanism will no longer be effective if a HIT exists. Fig. 3(a) presents the workflow of the user task that triggers a HIT. First, the user task and the interrupt handler are loaded into the user zone. After being authorized by the OS, the user task begins. At the beginning, the task is prohibited from accessing the system zone by the OS. Later the task triggers the Trojan, and a hidden interrupt emerges. As a privileged interrupt, it suspends the OS running on the processor, and directs the program counter to the address of the interrupt handler. Then the interrupt handler can access and modify the system zone as the OS does not work.

Assume an OS is running on the miniMIPS processor, an interrupt caused by HIT can actually suspend the OS. After the Trojan triggers the interrupt, the interrupt is sent to the coprocessor, and the coprocessor provides the address of the interrupt handler to the PF pipeline (the address is directed to the user zone). It is worth emphasizing that the PF pipeline preferentially selects the address of the interrupt handler as next program counter. Hence, the interrupt suspends the current normal program flow (including OS), and force the program counter to that address. In this case, this program at that address has read and write permissions to the full address space, or even can tamper the OS. Therefore, the proposed HIT can bypass the protection mechanism of the OS.

### C. The threat model of HIT

All processes of hardware design and manufacturing can be implanted in Trojans. In addition to untrusted IP providers and manufacturing foundries, chip design companies can also be compromised. Malicious modification to hardware in the design process poses a significant threat today [7]. On the one hand, the chip design company does not disclose all the functions of a chip to users, such as hidden privilege interrupts. On the other hand, for economic and military purposes, chip companies may implant backdoors into the products for users in specific organizations and regions. Developing the hidden hardware Trojan is a feasible way to implement such backdoors in the design process.

The program at the address of the interrupt handler gains super privileges, and enables normal users to steal key data in a computer, such as the password of the root user. Fig. 3(b) presents an example of the threat mode of HIT. In an OS, the passwords are usually stored in the system zone (e.g., /etc/shadow in Linux). The users except of the root user do not have read access. A hacker can log into the computer system as a normal user. He then loads the boot sequence and the illegal instruction (described in machine codes) into the user zone as a trigger for the HIT. Meanwhile, he places a program with copy function at the address of the interrupt handler.




(a) System with HIT (b) Intrusion behavior of HIT

Fig. 3. The attack mode and threat mode of the hidden instruction Trojan.

Since the interrupt handler address is already directed to the user zone, the above operation is feasible. The hacker then executes the trigger and generates an interrupt. The interrupt will suspend the current OS and force the processor to execute the copy program. At this point, the copy program is not regulated by the OS, and has full read and write access to the address space. Therefore, the copy program copies the root password to the user zone. Finally, the hacker can log in as the root user using the root password.

## III. The Optimization of Hidden Instruction Trojan

In this section, we use circuit reusing and integer programming to optimize the HIT, and make the Trojan to achieve low trigger probability, limited area, and insignificant power characteristics.

### A. Circuit reusing

The FSM of the HIT can reuse the decoder output signals to identify instructions in the boot sequence. Assume the processor contains $n$ instructions, denoted as $I_1$, $I_2$, …., $I_n$, and the decoder has $m$ output signals. The decoder provides a unique value of output signals for each instruction. Let $L_i$ is the value of minimum output signals that can distinguish the $i^{th}$ instruction from other instructions. Let $S_{Li}$ be the size of the circuit to identify $L_i$. If $L_i$ contains only a single output signal, $S_{Li}$ would be zero (or the area of a reverse gate).

Let $op_i$ and $S_{opi}$ be the opcode of the $i^{th}$ instruction and the size of the circuit to identify the opcode, Equation (1) shows the final area overhead (denoted as $S_i$) to identify the $i^{th}$ instruction. If $S_{Li}$ is less than $S_{opi}$, reusing the decoder can reduce the area overhead to identify the $i^{th}$ instruction.

$$S_i = \begin{cases} S_{Li} & S_{Li} < S_{opi} \\ S_{opi} & else \end{cases} \quad (1)$$

### B. Integer programming

We use integer programming [8] to minimize the trigger probability of the Trojan within a given area overhead. First, the probability that normal programs trigger the Trojan is determined mainly by the boot sequence. We assume the trigger probability of the boot sequence is the cumulative product of occurrence probability of its instructions because the boot sequence is immune to the effects of jump instructions as mentioned in Section II.A.

Let $P_{FSM}$ and $P_{Illegal}$ be the trigger probability of the boot sequence and the occurrence probability of the illegal instruction, respectively, Equation (2) calculates the trigger probability of the Trojan ($P_{Trojan}$). Assume the sequence of normal instructions $I_{b1}$, $I_{b2}$, ..., $I_{bt}$ is the boot sequence, $t$ is the sequence length, and the occurrence probabilities of these instructions are $P_{b1}$, $P_{b2}$, ..., $P_{bt}$. Equation (3) calculates the trigger probability of the boot sequence.

In addition, we assume the instructions in the boot sequence are different from each other to improve the concealment of the Trojan.

$$P_{Trojan} = P_{FSM} \times P_{Illegal} \quad (2)$$

$$P_{FSM} = \prod_{i=1}^{i=t} P_{bi} \quad (3)$$

Second, the area overhead of the instruction Trojan should be seriously reduced because the gap area of a chip is limited and a large Trojan can be easily discovered. Let $S_{FSM}$ and $S_{Illegal}$ be the circuit areas to identify the boot sequence and the illegal instruction, respectively, Equation (4) calculates the area overhead of the Trojan $S_{Trojan}$. In this work, we assume the processor recognizes an illegal instruction just by its instruction code, so $S_{Illegal}$ is a constant.

$$S_{Trojan} = S_{FSM} + S_{Illegal} \quad (4)$$

The FSM of a boot sequence contains two parts of the circuits: one for state transition and the other for identifying instructions in the boot sequence. Let $S_{State}$ and $S_{Inst}$ be the areas of these circuits, Equation (5) calculates the area overhead of the FSM. Assume the designer has optimized the FSM of the boot sequence for any given length $t$. That means $S_{State}$ is fixed for a given length $t$, and can be calculated by the function in Equation (6).

$$S_{FSM} = S_{State} + S_{Inst} \quad (5)$$

$$S_{State} = Fun(t) \quad (6)$$

Then $S_{Inst}$ is equal to the sum of the circuit area for identifying each instruction. Let $S_{bi}$ be the circuit area for identifying the $bi^{th}$ instruction in the boot sequence, we calculate $S_{Inst}$ using the Equation (7). Reusing the decoder for identifying most of the instructions is feasible, but the circuit area to identify each instruction varies greatly. Therefore, selecting the instruction with a small identifying circuit can reduce the area overhead of the HIT.

$$S_{Inst} = \sum_{i=1}^{i=t} S_{bi} \quad (7)$$

Assume the maximum area overhead for the Trojan is $S_{Baseline}$, we model the problem of optimizing the HIT as follows:

Input: { $P_1$, $P_2$, ..., $P_n$ }, { $S_1$, $S_2$, ..., $S_n$ }, and $Fun(t)$
Constraint: $S_{Trojan} < S_{Baseline}$
Output: { $I_{b1}$, $I_{b2}$, ..., $I_{bt}$ }
Target: Minimize ($P_{Trojan}$)

Integer programming [8] can solve this problem, generate the boot sequence of length $t$, and minimize the trigger probability. In addition, $P_{bi}$ and $S_{bi}$ in the previous Equations belong to the inputs {$P_1$, $P_2$, ..., $P_n$ }and {$S_1$, $S_2$, ..., $S_n$}, respectively, and $P_{Illegal}$ is set as the ratio of a single opcode to all possible opcodes for simplicity.

### C. Optimizing the instruction Trojan in miniMIPS

Since the proposed problem is nonlinear, we apply the Monte Carlo method [9] to solve the problem. Fig.4 presents the algorithm to optimize the HIT. First, the algorithm randomly generates a set of integer variables to represent a boot sequence. Then, it checks if the boot sequence meets the constraint of area overhead. If yes, the algorithm goes to the next step; else, it discards the boot sequence. Later, the algorithm calculates the trigger probability of the boot sequence. If the trigger probability is less than that of the optimal solution, the algorithm updates the optimal solution with the current boot sequence; else, it discards the boot sequence. Finally, if the loop *num* exceeds the loop threshold, the algorithm ends up; else, it continues. By setting a large threshold, Monte Carlo can solve the nonlinear problem, and provide an approximately optimal boot sequence to minimize the trigger probability for the Trojan.

The Monte Carlo method can find an approximately optimum boot sequence that minimizes the trigger probability for the Trojan.

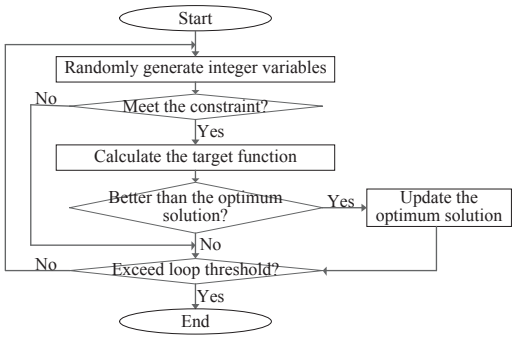
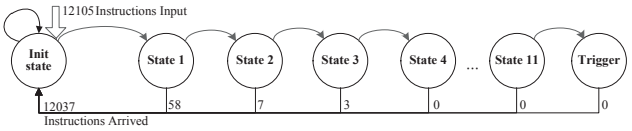



Fig. 4. The algorithm for optimizing the intruction Trojan.




(a) The number of each state triggered by the random programs.

| Type | ITMAT | OVERF | ERINS | BREAK | SCALL | HIT Interrupt |
|---|---|---|---|---|---|---|
| Number | 0 | 11 | 4 | 2 | 1 | 0 |

(b) The types and number of the activated interrupts.

Fig. 5. The simulation results on the processor with HIT.

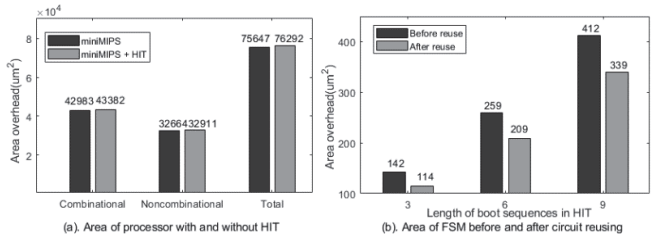



Fig. 6. The area overhead of the FSM before and after circuit reusing.

Usually, the gap area in a chip is more than 1% chip area, so we assume the area threshold of the HIT is 0.8% processor area. Then we gradually increase the loop threshold and execute the Monte Carlo simulation to obtain a new minimized solution. When the loop threshold reaches one million, the trigger probability no longer decreases. Even if the loop threshold reaches ten million, the trigger probability remains the same. In this case, we believe that the algorithm obtains an almost optimum solution. The boot sequence of this solution consists of 11 instructions, and the trigger probability by normal programs is as low as $10^{-40}$. Therefore, normal programs cannot trigger this Trojan. The instruction Trojan can be deeply hidden inside the processor, and survive from the detection of the simulation-based methods.

## IV. Experiment Results

In this section, we implant the optimized HIT into the miniMIPS processor, and use the Trojan-free one as a comparison. First, we synthesize two processors using a 90 nm lib, and obtain their netlists and area overhead. Later, we prepare four random programs, which are compute-intensive, jump-intensive, memory-intensive, and control-intensive programs, respectively. Among them, the control-intensive program is a random program that can activate the pipeline control logic. Next, we execute these programs on the two processors and check if these programs trigger the HIT. Finally, we obtain signal flipping ratios from the simulation tool and analyze their power characteristics.

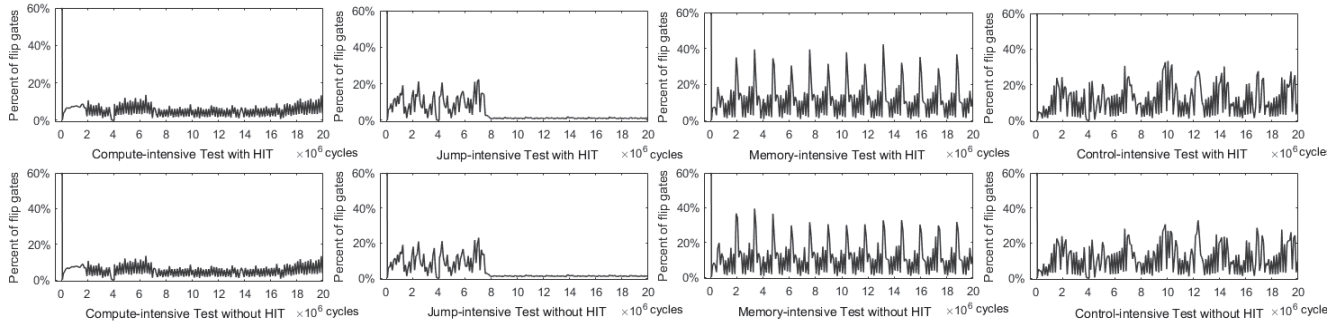



Fig. 7. The gate flip ratio of the processors with and without HIT.

### A. Trigger probability by normal programs

The integer programming algorithm significantly reduces the probability that the Trojan is triggered by normal programs. Fig. 5(a) shows the triggered times of each state in the FSM when the processor with HIT executes four random programs. The programs contain more than 10000 instructions in total, and cover most functions of the miniMIPS processor, but they only trigger a small portion of the FSM states. In Fig. 5(a), most instructions just make the FSM stay at the *init* state. Only 58 instructions force the FSM into *state*1. The number of instructions that change the FSM from *state*1 to *state*2 quickly falls to seven, and only three instructions make the FMS enter *state*3. These programs have never triggered the remaining FSM states. Even if we massively increase the size of random programs, activating all FSM states remains difficult. That is because we choose the low occurrence-probability instructions, increase the length of the boot sequence, and thus reduce the trigger probability to an extremely low value. The simulation results demonstrate that the optimized boot sequence improves the concealment of the instruction Trojan.

The boot sequence with a low trigger probability hides the new interrupt of the HIT. We insert four illegal instructions into the random programs, and two of them are just the illegal instructions in the Trojan. Fig. 5(b) presents the types and numbers of interrupts that occur during the simulation. In the simulation process, even if the illegal instruction of the Trojan emerges, it just results in an ERINS interrupt instead of causing the hidden privileged interrupt. Because the proposed HIT requires the entire boot sequence to appear before the illegal instruction. The low triggered probability of the boot sequence ensures that the HIT can survive from the detection of the simulation-based methods.

### B. Area overhead and power characteristics

The hidden instruction Trojan can be smoothly implanted into the gap area of the processor. First, the integer programming algorithm effectively constrains the area overhead of the Trojan. In Fig. 6(a), the processor with HIT increases 0.85% total area, slightly more than the area threshold of 0.8%. Second, reusing the decoder of the processor effectively reduces the area overhead of the instruction Trojan. In Fig. 6(b), reusing the decoder effectively reduces the circuit area of the FSM regardless of the boot sequence length. The low area overhead makes the instruction Trojan can be implanted in processors easily.

The optimized HIT has insignificant power characteristics. Although all parts of the processor are activated and flipped by these random programs, the average flipping ratio of the gates in the processor with HIT is basically the same to that in the processor without HIT. As shown in Fig.7, the lines of the gate-flipping ratios in two processors are the same in both frequency and amplitude. A few slight differences exist, but the magnitudes of these differences are just within the noise level. That is because the Trojan area is strictly limited, and the power brought in by the HIT is quite little. Meanwhile, the Trojan circuits (e.g., the mask) flips synchronously with input instructions instead of keeping quiet, hence the power characteristic is insignificant. That ensures the Trojan can survive from the detection of the side-channel signal analysis.

## V. Conclusions

In this paper, we model a typical hidden instruction Trojan (HIT). The Trojan uses a low-probability instruction sequence as a boot sequence and is triggered by a followed illegal instruction. The payload of the Trojan is a hidden interrupt to force the program counter to a specific address and make the program at the address with super privilege. Finally, the Trojan is optimized by the integer programming algorithm to minimize the trigger probability within a given area overhead. The experimental results demonstrate that HIT has an extremely low trigger probability and can survive from the detection of the existing test methods.

## References

[1] M. Tehranipoor and F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," IEEE Design & Test of Computers, vol. 27, no. 1, pp. 10-25, Jan.-Feb. 2010.

[2] I. H. Abbassi, F. Khalid, S. Rehman, A. M. Kamboh, A. Jantsch, S. Garg, and M. Shafique, "TrojanZero switching activity-aware design of undetectable hardware Trojans with zero power and area footprint", in Proc. Design, Automation & Test in Europe, Florence, Italy, 2019.

[3] N. Fern and K. Cheng, "Detecting hardware Trojans in unspecified functionality using mutation testing," in Proc. IEEE/ACM International Conference on Computer-Aided Design, Austin, TX, 2015, pp. 560-566.

[4] W. Zhao, H. Shen, H. Li and X. Li, "Hardware Trojan Detection Based on Signal Correlation," in Proc. Asian Test Symposium, Hefei, 2018, pp. 80-85.

[5] X. Wang, Q, Zhou, Y. Cai, and G. Qu, "Parallelizing SAT-based de-camouflaging attacks by circuit partitioning and conflict avoiding", Integration, vol.67, pp.108-120, 2019.

[6] J. Zhang, F. Yuan, L. Wei, Y. Liu, and Q. Xu, "VeriTrust: Verification for hardware trust", IEEE Transactions on CAD of Integrated Circuits and Systems, Vol.34, No.7, 2015, pp.1148-1161.

[7] A. Waksman, S. Sethumadhavan, "Silencing Hardware Backdoors", in Proc. IEEE Symposium on Security and Privacy, Berkeley, CA, 2011, pp. 22-25.

[8] M. P. Wellman; A. Greenwald; P. Stone, "Integer Linear Programming Formulations," in Autonomous Bidding Agents: Strategies and Lessons from the Trading Agent Competition, MITP, 2007, pp.

[9] A. Z. Grzybowski and P. Puchala, "Monte Carlo simulation of the Young measures — Comparison of random-number generators," in Proc. IEEE International Scientific Conference on Informatics, Poprad, 2015, pp. 109-113.