Combating Enhanced Thermal Covert Channel in Multi-/Many-core Systems with Channel-aware Jamming

Jiachen Wang, Xiaohang Wang, Yingtao Jiang, Amit Kumar Singh, Letian Huang and Mei Yang

***Abstract*—As a means to thwart thermal covert channel attack in a multi-/many-core system, a strong heat noise whose frequency band coincides with that occupied by the thermal covert channel is injected to jam the channel. However, this undiscriminating channel jamming based countermeasure will fail if a thermal covert channel is allowed to change its transmission frequency dynamically in response to the jamming. To combat this enhanced thermal covert channel, a more advanced countermeasure is needed and thus proposed that checks the frequency spectrum and tracks any possible covert channel. Only after a channel is detected to be susceptible, a thermal noise with this channel frequency is then emitted to jam the covert channel. The communication protocols and frequency changing scheme pertaining to this enhanced thermal covert channel are described in this paper. The experimental results confirm that, when the proposed countermeasure is applied, the enhanced thermal covert channel, much more resilient to jamming, suffers from an extremely high packet error rate (PER), which makes any meaningful data leakage practically impossible. As the proposed countermeasure method is poised to contain dangerous thermal covert channel attacks with an anti-jamming capability, it lends itself well to secure multi-/many-core systems.**

***Index Terms*—Thermal Covert Channel, Signal Detection, Defense against Covert Channel Attack, Many-core Systems.**

I. Introduction

EXISTENCE of covert channels in multi-/many-core chips can cause catastrophic information breach. Of many different communication media available for the construction of covert channels [1]–[13], chip temperature (thermal) remains the most widely used, as a thermal covert channel can secretly, reliably transmit data in a "wireless" manner. In [2], for instance, it was demonstrated over two hardware platforms featuring quad-core Intel Core i7-4710MQ processor and Samsung Exynos 5422 SoC that a thermal covert channel can transmit secret data, like passwords, at a rate of 5 bits per second (bps). Actually, transmission rates of a covert channel can go up to 20 bps and even higher, as indicated in [2], [3]. In general, when the transmitter of a processor core tries to send out supposedly secret data over a thermal covert channel, it converts data streams to temperature signals for transmission. The receiving processor on the other end reads the thermal signal from its embedded thermal sensor and then converts the signal back to data [1].

As a countermeasure against thermal covert channel attack, a system can run specific programs with a sole purpose of generating thermal noise that is strong enough to block any possible communications over the said channel. Ideally, the generated thermal noise spans the entire frequency spectrum at all times to ensure a complete channel jam. This wide spectrum, always-on jamming strategy, however, consumes significant amount of energy to keep the thermal channel jammed over even a modest duration of time. A more subtle method requires jamming only at the same frequency band occupied by the thermal covert channel, preferably at the time when the channel is being used for data transmission.

Although targeted jamming can be an effective countermeasure against a thermal covert channel, one can see that this scheme can become less effective, or nearly useless, should the thermal covert channel be able to dynamically change its transmission frequency, as illustrated in Fig. 1. Assume the covert channel in this example (Fig. 1) occupies the transmission frequency of 40Hz, and a program running on another processor is dedicated to continuously monitor any possible covert channel attacks by reading its own thermal sensor. Once a signal is detected and determined to be part of covert channel communication, strong thermal noise centered at 40Hz is triggered (Fig. 1(a)). By injecting such a strong thermal noise, the signal-to-noise-ratio (SNR) of the receiver deteriorates and the packet error rate (PER) escalates to a very high level (89%) at which any data communication is essentially shut down. Since both the transmitter and the receiver find their PERs exceed the level that makes them believe channel jamming is taking place, the covert channel switches to the transmission frequency of 60Hz in Fig. 1(b). An immediate result is that the PER drops to just 4%. This example (Fig. 1(b)) clearly indicates that a thermal covert channel can become more resilient, and thus more harmful, provided it is given the extra ability to dynamically change its transmission frequency. In this paper, we first elaborate on an anti-jamming thermal covert channel which is able to avoid

Manuscript received April 18, 2020; revised June 12, 2020; accepted July 6, 2020. This article was presented in the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems 2020 and appears as part of the ESWEEK-TCAD special issue.

J. Wang is with the School of Software Engineering, South China University of Technology, Guangzhou, Guangdong, 510006, China. E-mail: jiachen wang3@gmail.com.

X. Wang is with the School of Software Engineering, South China University of Technology, Guangzhou, Guangdong, 510640, China. He is the corresponding author. E-mail: xiaohangwang@scut.edu.cn.

Y. Jiang and M. Yang are with the Department of Electrical and Computer Engineering, University of Nevada, Las Vegas, NV89557. E-mail: ying tao.jiang@unlv.edu, mei.yang@unlv.edu.

A. K. Singh is with the University of Essex, UK. E-mail: a.k.singh@essex.ac.uk

L. Huang is with the University of Electronic Science and Technology of China, China. E-mail: huanglt@uestc.edu.cn

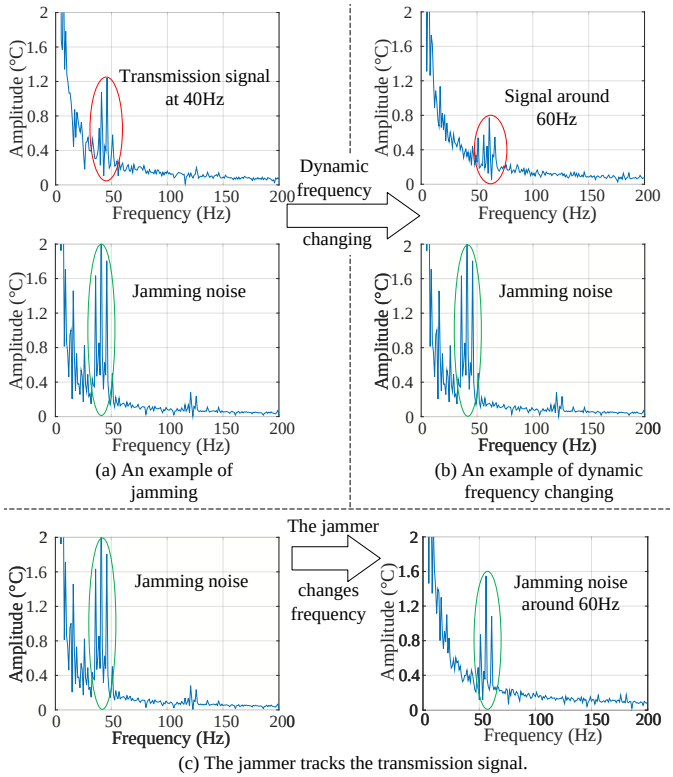

Fig. 1. (a) Jamming of a thermal covert channel (PER=89%). (b) Using the dynamic frequency changing method to avoid the channel being jammed (PER=4%). (c) The jammer changes its frequency dynamically to track the thermal signal transmission (PER=86%).

being jammed.

However, if the jammer is able to track the transmitter's frequency and insert the jamming noise accordingly, as the case shown in Fig. 1(c), it switches the noise frequency also to be centered around 60Hz; this way, the covert channel attack is blocked with a high PER of 86%. This example thus has motivated us to propose a countermeasure scheme based on periodically scanning the frequency spectrum to detect any possible attack. Once a covert channel is detected and its transmission frequency is determined, a strong noise source of falling into the same transmission frequency band is applied for targeted jamming. We demonstrate in this paper that such jamming method can completely block any thermal covert channel that otherwise can be exploited for data leakage.

The remainder of the paper is organized as follows. Section II reviews the related work about covert channel and challenges when the traditional frequency hopping technique is mixed with thermal covert channel. Section III defines a dynamic frequency changing protocol for thermal covert channel. Section IV details the countermeasure method. Section V presents the experimental results. At last, Section VI concludes this paper.

II. Related Work

A thermal covert channel transmits sensitive data among cores through thermal signal. In an experiment reported in [1] on a real platform, a thermal covert channel was shown to produce a throughput of 1.33 bits per second (bps) and a bit error rate (BER) of 11%. Bartolini *et al*. [2] changed the encoding scheme of thermal covert channels and achieved an even lower bit error rate of 0.1% at 8 bps. Long *et al*. [3] considered the influence of the noise generated by the other applications running on the system. They used high signal frequency thermal covert channel to avoid noise.

The thermal covert channel can be divided into three layers: the *physical layer*, the *link layer* and *application layer*.

The *physical layer* in thermal covert channel is responsible for the conversion between digital/electronic and temperature signals. Digital signal bit streams '1' s and '0' s are converted by temperature levels (*e.g.*, high or low temperatures) or variations. The temperature signals propagate from the transmitter to the receiver by means of heat transfer, which can be modelled by Hotspot [14]. On the receiver side, it reads its thermal sensor to convert the thermal signal back to digital.

Note that temperature sensors are essential for dynamic thermal management [15], [16] and they are widely deployed in today's chips. The number and accuracy of temperature sensors are possible to be ever improved in the future [17]. For programs, instantaneous temperature can be retrieved by software interfaces with a resolution of 1 C [18], and the state-of-the-art thermal sensor has a resolution close to 0.1 C [19]. After retrieving temperature signal, it passes through filters, after which only the frequency components of interest remain.

The *link layer* is responsible for connecting the transmitters and receivers. At the link layer, digital signals taken from the physical layer are decoded, and the bits are combined into packets for transmission. [1] and [2] used a simple protocol that transmit raw data without control packets, while [3] used control packets to identify the beginning and the end of a communication session.

In the *application layer*, a hacker designs how to encode the sensitive data using the thermal covert channel. Popular coding standards like UTF-8 and ASCII can be used for this purpose.

Frequency-Hopping Spread Spectrum (FHSS) can improve the anti-jamming ability of wireless communication [20]. In FHSS, the transmission frequency changes from time to time to avoid being jammed. Both the transmitter and receiver maintain the same frequency hopping pattern. Several schemes were proposed to synchronize the transmitter and receiver. For example, in [21], a dedicated channel is used to transmit synchronization information. After receiving the synchronization information sent by the transmitter from this dedicated channel, the frequency hopping pattern, frequency sequence, and start and end times of the receiver are set according to the instruction of the synchronization information. The self-synchronizing method [22] relies on the synchronization information extracted from the received frequency hopping signal to achieve frequency hopping synchronization. The synchronization-head method [21] selects one or more channels to transmit a special set of packets carrying synchronization information before frequency-hopping communication. After receiving the synchronization information packet, the receiver performs clock calibration and frequency hopping according to the instruction of the synchronization information.

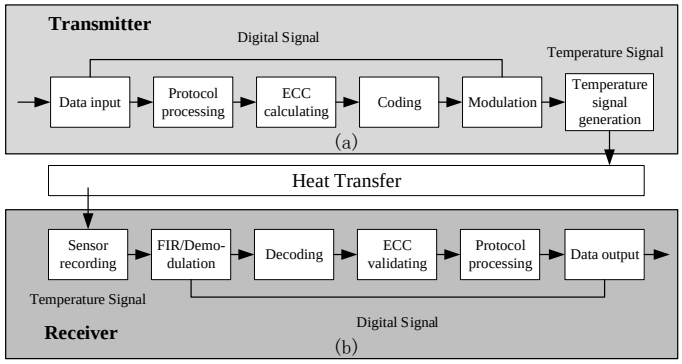

Fig. 2. The flow of a baseline thermal covert channel transmission from the (a) transmitter end to the (b) receiver end.

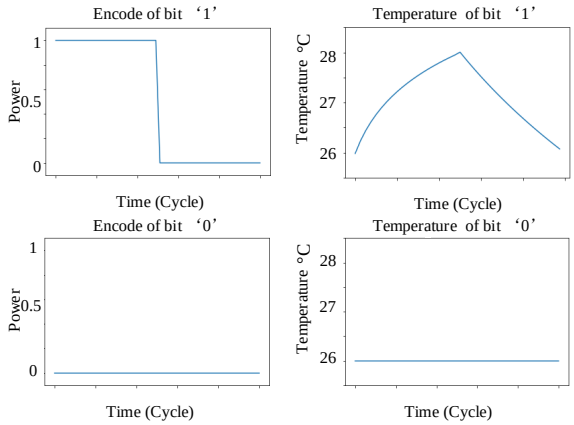

Fig. 3. The encoding scheme used in the baseline thermal covert channel.

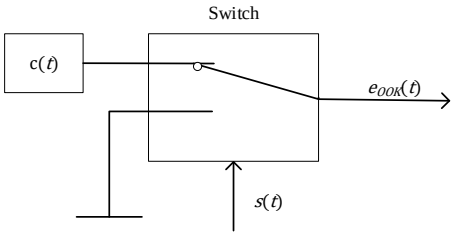

Fig. 4. The generation of the OOK signal.

However, these FHSS systems cannot be applied directly to the thermal covert channel due to their huge implementation overhead. For the dedicated channel scheme, this dedicated channel might also be jammed, leading to failure of the whole FHSS system.

III. Dynamic Frequency Changing Protocol Based Thermal Covert Channel

In this Section, we propose a lightweight scheme to dynamically change the transmission frequency with a very low implementation overhead, and without involving any extra channel. In essence, we use a polling based frequency changing protocol and the available transmission frequencies are stored by both the transmitter and receiver in advance. In the worst case that the channel is severely jammed, both the transmitter and the receiver poll the next available frequency iteratively and send a series of packets as an attempt to set up their connection over that channel. Once they find a channel available for connection, communication resumes in this new channel.

A. Baseline Thermal Covert Channel Model

The baseline thermal covert channel [3] links a transmitter and a receiver, as shown in Fig. 2. The transmitter and receiver run on different cores or on different hardware threads of a physical core if the processor supports multi-threading. The transmitter sends the sensitive data via the thermal covert channel. The receiver records the temperature signal by reading its thermal sensor, and decodes the signal to recover the original data.

1) The Transmitter: It includes the following modules:

(1). **The data input module** that reads the data to be transmitted. The sensitive data are binary streams. For example, a password letter 'A' may be encoded as a bit stream of "0100 0001" in ASCII code.

(2). **The protocol processing module** that generates the control bits of a packet, as required by the communication protocol.

(3). **The Error Correcting Code (ECC) calculating module** that gets the data stream with protocol information and calculates the corresponding error correcting code. The ECC code is tail-added to a packet.

(4). **The coding module** that encodes the data to improve the communication reliability. It uses the return-to-zero (RZ) line coding scheme; that is, it encodes bit '1' by a rise and fall in temperature level, and '0' by consecutive low temperature level. The encoding scheme avoids the continuous buildup of temperature. This encoding scheme is illustrated in Fig. 3.

(5). **The modulation module** that uses OOK (On-Off Keying) which is shown in Fig. 4 to modulate the binary stream in the packet. The signal is non-zero when the bit is '1' and becomes zero when the bit is '0'. The OOK signal can be expressed as

$$e_{OOK} = s(t) \quad c(t) \tag{1}$$

$$s(t) = a_n g(t \quad nT_B) \tag{2}$$

where $c(t)$ is the carrier signal, $s(t)$ is the signal of packet after ECC encoding, T_B is the period of one bit (symbol width), $g(t)$ is the baseband pulse waveform with duration T_B, and a_n is the value of the n-th bit of the packet to be transmitted ('0' or '1').

(6). **The temperature signal generation module** that is a program generating temperature signals by controlling the power consumption of the core according to the modulated data packets.

2) The Receiver: It includes the following modules:

(1). **The sensor recording module** that reads the temperature information through the system APIs or calls **rdmsr** instruction for data from model specific registers (MSR) in the processor (contains instantaneous temperature information). These APIs or instructions can be accessed in user space. This module records temperature trace.

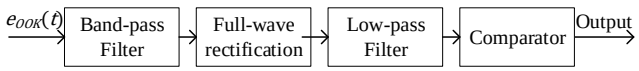

Fig. 5. The demodulation of OOK modulated signal.

(2). **The FIR module** that uses a few finite impulse response (FIR) filters with the center frequency tuned to the transmission frequency of the covert channel to filter the thermal signal. This module has a band-pass filter, a full-wave rectification, and a low-pass filter connected in series, as shown in Fig. 5.

(3). **The decoding module** that uses the result of FIR filter and compares the amplitude of signal with a decision threshold T_b to make a hard decision about the binary value. There are two options for the decision threshold T_b. One is to use a *fixed threshold* which is selected to be the maximum amplitude of noise profiled at offline. Another is a *dynamic threshold*, which is the half amplitude of the thermal covert channel's thermal signal. The output of this module is a binary bit stream.

(4). **The ECC decoding module** that decodes the ECC code of the data and checks the data integrity, followed by computing the error rate.

(5). **The protocol processing module** that reads the content of the packet and interpret the protocol control and data bits.

(6). **The data output module** that writes the data to a buffer. For example, password letter 'A' may be saved as a string.

3) The Communication Protocol of Transmitters and Receivers in the Baseline Thermal Covert Channel: The communication protocol is defined as follows.

Step 1. The transmitter sends a request packet (REQ) to the receiver and waits for its response.

Step 2. Once the receiver receives the REQ, it replies the transmitter with an acknowledgement packet (ACK).

Step 3. The transmitter sends DATA packets to the receiver.

Step 4. The receiver receives each packet and decodes the ECC code to check whether the packet is compromised or not. If not, the receiver replies an ACK packet in the same channel. Otherwise, it sends nothing back.

Step 5. The transmitter sends a terminate packet (TER) to the receiver to terminate the transmission after all the packets are sent.

B. Communication Protocol to Support Dynamic Frequency Changing

Suppose the covert channel can use a set of transmission frequencies $F = \{f_1; f_2; \ldots; f_m\}$. In the beginning, the transmitter and receiver are initialized to use the same frequency f_1. They communicate in a regular manner, as described in Section III-A.

There are three mechanisms to detect whether a channel is being jammed or not.

The first one is referred as self-checking by the transmitter. When the transmitter is sending, it receives packets from its temperature sensor simultaneously. The packet received is compared with the sent packet to compute the error rate. If the bit error rate exceeds a threshold T_{BER}, the channel is considered being jammed. Comparing with the approach that calculates the bit error rate in the receiver, self-checking in the transmitter has a higher degree of accuracy because the original packet is known to the transmitter.

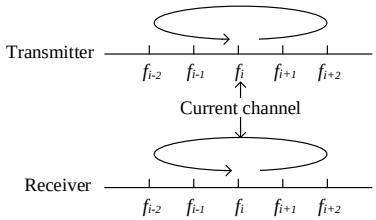

Fig. 6. Polling of the transmitter and receiver in a round robin order.

The second mechanism requires the transmitter keep a failure accumulation counter C_f to count the number of accumulated failures observed by the transmitter. When a packet is transmitted, the transmitter starts a timer T_p. If there is no reply from the receiver before T_p expires (exceeds T_1), C_f increases by 1. If the reply arrives on time, C_f is reset to 0. Once the failure accumulation counter exceeds a threshold of 3, the channel is considered being jammed.

Though the two mechanisms described above are able to deal with most jamming situations, in some extreme conditions, the dynamic frequency changing request packets (DFCQ) from the transmitter cannot be correctly decoded by the receiver. This can occur when the jamming noise is very strong. Another possibility is when the transmitter and the receiver are running at different frequencies, and they are not able to communicate with each other. To solve this problem, the third mechanism, polling, as shown in Fig. 6, is needed (lines 30-32 in **Algorithm 1** and lines 16-18 in **Algorithm 2**). Both the transmitter and the receiver set a timer T_{tcc} when last correctly received packet arrives at f_i. If T_{tcc} expires (exceeds threshold T_2), both the transmitter and the receiver switch their frequencies following a pre-agreed round robin order $\{f_1 \to f_2 \to \ldots \to f_m \to f_1 \to f_2; \ldots\}$ without talking to each other. The transmitter changes its transmission frequency to f_{i+1} and transmits its DATA packets. If no ACK packets are received after some time, the transmitter changes its transmission frequency to $f_{i+2}; f_{i+3}; \ldots$ iteratively after a time interval of T_3, until an ACK packet is received. The receiver changes its receiving frequency to f_{i+1} and prepares to receive packets at f_{i+1}. If no correct packets are received, the receiver changes its frequency to f_{i+2}. Finally, the transmitter and the receiver converge to a new frequency.

Once being jammed, dynamic frequency changing is triggered. The communication protocol for dynamic frequency changing works as follows. Assume now the covert channel works at frequency f_i.

1. The transmitter checks every frequency in the available transmission frequency set F excluding f_i which is currently being used. For each selected frequency f_j, the transmitter tests whether that channel is being jammed or not by computing the BER. If f_j is not available, the next frequency is selected and tested iteratively until an available frequency is found. If f_j is available, dynamic frequency changing starts.
2. The transmitter sends a dynamic frequency changing

Algorithm 1 The transmitter process

Input:
$Data$: n bits of the data to be transmitted
F: the available frequencies of thermal covert channel $F = \{f_1, f_2, ..., f_m\}$
f_i: the channel frequency that the transmitter uses at the beginning

```
1: while Data is not empty do
2:     /* Normal communication */
3:     Fetch bits from Data and assemble them to form a packet
4:     Transmit the packet and self-checking at f_i
5:     Turn on the timer T_p
6:     Wait for a reply from the receiver
7:     if T_p expires then
8:         Retransmit the packet
9:     end if
10:    /* Dynamic frequency changing(DFC) trigger */
11:    /* Failure accumulation counter and self-checking */
12:    if C_f >3 or self-checking error then
13:        DFC is triggered
14:    end if
15:    /* Dynamic frequency changing */
16:    if DFC is triggered then
17:        while DFC is not finished do
18:            Select an available frequency f_j from F
19:            Transmit DFCQ(f_i,f_j)
20:            Wait for DFCA(f_i,f_j)
21:            if DFCA(f_i,f_j) received then
22:                f_i  f_j
23:                DFC finish
24:            end if
25:            if T_p expires then
26:                Retransmit the packet
27:            end if
28:        end while
29:    end if
30:    if T_tcc expires then
31:        Change the transmission frequency in the round robin order of {f_1,f_2,...,f_m,f_1,f_2,...} until a communication frequency is found
32:    end if
33: end while
```

request DFCQ(f_i, f_j) to the receiver, indicating that this packet will be sent at f_i (which is the current thermal covert channel working frequency) and requests to change to f_j. The transmitter then sets a packet timer T_p for the DFCQ packet and waits for the reply from the receiver. If the reply is received before T_p expires, the transmitter goes to step 4 otherwise, DFCQ is retransmitted.

3. The receiver performs the following steps after receiving the DFCQ(f_i, f_j) packet:

   3.1. It sets the center frequency of the FIR filter to be f_j after receiving the request packets.

   3.2. It sends a dynamic frequency changing acknowledgement DFCA(f_i, f_j) to the transmitter at f_i, indicating that the covert channel now works at f_i and agrees to change to f_j.

   3.3. The receiver sets a dynamic frequency changing timer T_{fh}. T_{fh} expires if it exceeds a threshold T_4. If the receiver receives packets at the new frequency f_j, which means the transmitter also changes to f_j successfully, this timer is reset. If T_{fh} expires, the receiver starts up the polling mechanism.

4. If the transmitter receives the DFCA(f_i, f_j) packet, it changes its transmission frequency to be f_j and self-checks frequency f_j. It starts to send the data packets at the new frequency f_j. If no packet is received before T_p expires, the transmitter goes back to step 2.

1) The Transmitter: The transmitter includes a transmitting module, an ACK/NACK analyzing module for error rate computation, a dynamic frequency changing trigger, and a dynamic frequency changing module, as shown in Fig. 7.

The transmitting module that sends sensitive data via a thermal covert channel.

The ACK/NACK analyzing module that records the temperature signal by reading its thermal sensor, and decodes the signal to be ACK or NACK.

Algorithm 2 The receiver process

Input:
F: the available frequencies of thermal covert channel $F = \{f_1, f_2, ..., f_m\}$
f_i: the channel frequency that the receiver uses at the beginning
Output: $OutputData$: the data received from the transmitter

```
1: while transmission is in session do
2:     if a new packet is received then
3:         if the packet is a data packet then
4:             /* Normal communication */
5:             /* Save the data */
6:             Add the payload of packet into OutputData
7:             Reply ACK to the transmitter
8:         end if
9:         if the packet is DFCQ(f_i,f_j) then
10:            /* DFC is triggered */
11:            Reply DFCA(f_i,f_j) to the transmitter
12:            Change the receiving frequency to f_j
13:        end if
14:    end if
15:
16:    if T_tcc expires or T_fh expires then
17:        Change the receiving frequency in the round robin order of {f_1,f_2,...,f_m,f_1,f_2,...} until a communication frequency is found
18:    end if
19: end while
    Return OutputData
```

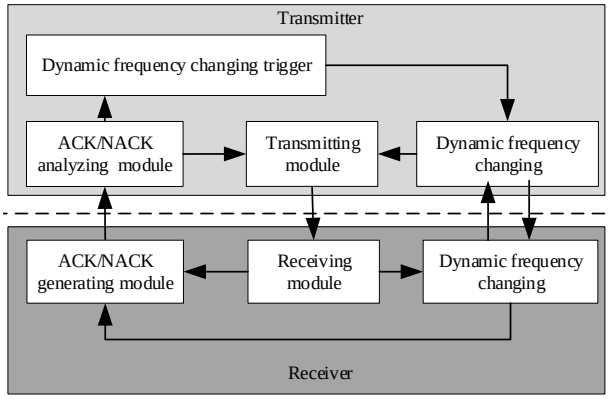

Fig. 7. Workflow of dynamic frequency changing in a thermal covert channel.

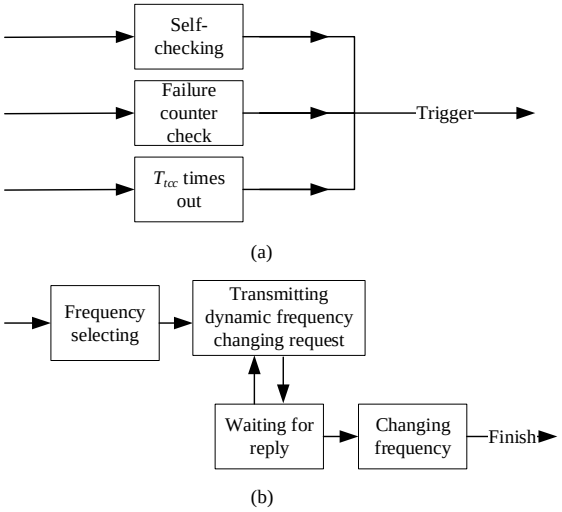

Fig. 8. (a) The dynamic frequency changing trigger. (b) The dynamic frequency changing module.

The dynamic frequency changing trigger module, shown in Fig. 8(a), is used to detect jamming. There are three mechanisms to detect jamming, self-checking, failure accumulation counter, and polling. The transmitter uses all the three detection mechanisms at the same time. Any one of them can trigger dynamic frequency changing.

The dynamic frequency changing module that is shown in Fig. 8(b). It is used to change to a new frequency to avoid being jammed. First it selects an available frequency to change to (corresponding to line 18 in **Algorithm 1**). Then the transmitter transmits DFCA (f_i, f_j) and waits for the reply from the receiver (corresponding to lines 19-24 in **Algorithm 1**). If there is no reply before T_p expires, the DFCQ (f_i, f_j) should be retransmitted. If a DFCA (f_i, f_j) packet is received from the receiver, the transmitter changes its transmission frequency to be f_j and communicates at f_j.

2) The Receiver: As shown in Fig. 7, the receiver includes a receiving module, an ACK/NACK generating module, and a dynamic frequency changing module.

The ACK/NACK generating module that sends the ACK/NACK packets via the thermal covert channel.

The receiving module that records the temperature signal by reading its thermal sensor, and decodes the signal.

The dynamic frequency changing module that sets the center frequency of the FIR filter to be f_j and replies a DFCA packet to the transmitter (corresponding to lines 9-13 in **Algorithm 2**).

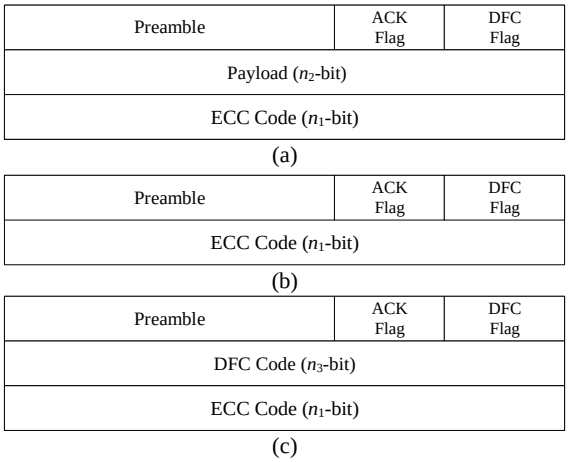

Fig. 9. Formats of (a) DATA, (b) DFCA and ACK, and (c) DFCQ packets that are in the dynamic frequency changing protocol.

C. Implementation of Thermal Covert Channel

1) Packet Format: Each type of packets has the following data fields: a preamble, an ECC code, an ACK and DFC flag. The preamble data field is a bit steam of "101010", which marks the beginning of a packet. A 1-bit ACK Flag is '1' if the packet type is ACK, and '0' otherwise. The 1-bit DFC Flag is set to be '1' if it is a DFCQ or a DFCA packet, and '0' otherwise. An n_1-bit ECC code is used for error correction. An n_3-bit DFC code is used for indicating a transmission frequency.

In our protocol, there are four types of packets, DATA, ACK, DFCQ and DFCA.

A DATA packet has a preamble, an ACK flag (set to be '0'), a DFC flag (set to be '0'), an n_2-bit payload, and an n_1-bit ECC code, as shown in Fig. 9(a).

An ACK packet has a preamble, an ACK flag (set to be '1'), a DFC flag (set to be '0'), and an n_1-bit ECC code, as shown in Fig. 9(b).

A DFCA packet has a preamble, an ACK flag (set to be '1'), a DFC flag (set to be '1'), and an n_1-bit ECC code, as shown in Fig. 9(b).

A DFCQ packet has a preamble, an ACK flag (set to be '0'), a DFC flag (set to be '1'), an n_3-bit DFC code, and an n_1-bit ECC code, as shown in Fig. 9(c).

2) Generation of the Temperature Signal: To generate the thermal signals, programs with CPU-intensive and CPU-idle codes are used to generate high and low temperatures. Temperature variation is modulated by controlling the power consumption of the processor. For the transmission frequency f_1, the period of signal variation is $1/f_1$. To transmit a bit '1', a CPU-intensive program is run in the first $1/2f_1$ period,

Algorithm 3 Generate temperature signal

Input:
Bit: 0 or 1
Period: the time that this process will run
Frequency: the frequency this process runs at

if Bit is 0 **then**
  Sleep for *Period*
else
  while Not exceed *Period* **do**
    Run for half of a *Period* at *Frequency*
    Sleep for half of a *Period* at *Frequency*
  end while
end if

followed by a CPU-idle program run in the next $1/2f_1$ period. To transmit a bit '0', the CPU-idle program is run for the whole $1/f_1$ period.

The code of temperature signal generation is showed in **Algorithm 3**.

D. Comparison with FHSS

The difference between the conventional FHSS and the proposed method are as follows. The proposed method detects the occurrence of jamming and then changes the transmission frequency, while the conventional FHSS needs additional communication for synchronization. When the additional signals/channels are jammed, the whole FHSS system does not work. In contrast, the attack in our approach can still work when the channel is jammed, as it periodically polls for the next available frequency channel. In addition, the conventional FHSS needs high computation effort for signal synchronization, which is not suitable for an on-chip covert channel.

IV. Countermeasure with Scanning and Channel-aware Jamming

To countermeasure the above thermal covert channel, an intuitive approach is to use a full-band jamming, meaning that thermal noise will flood the entire band $[f_l; f_h]$ occupied by the thermal covert channel. However, such a naive approach causes excessive power consumption, which might even lead to thermal overheating. Instead, a lightweight countermeasure as shown in Fig. 10 is proposed in this Section which periodically scans the frequency spectrum to check if there exists a potential thermal covert channel attack or not. If such a channel is identified, a noise is emitted with the same frequency of the covert channel. With a high scanning speed, all the thermal covert channels shall be able to be detected and jammed. The detection process is first introduced, followed by the jamming process.

A. The Detection Schemes

Two schemes can be used for the detection of a thermal covert channel. The temperature data are collected from the local temperature sensors.

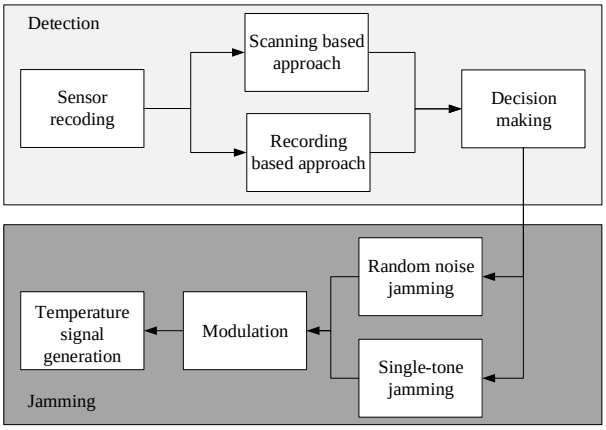

Fig. 10. Workflow of the proposed countermeasure.

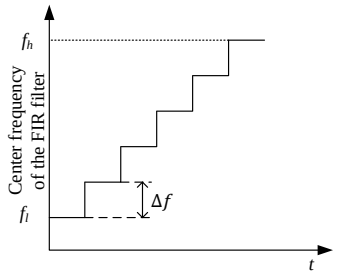

Fig. 11. Linearly scans the frequency band of $[f_l; f_h]$.

(1) **Scanning-based approach**. In this approach, the jammer changes the center frequency of the FIR filter to scan the entire frequency spectrum of interest. There are two different ways to scan the frequency spectrum.

A linear scan method changes the center frequency sequentially from the lowest frequency f_l to the highest frequency f_h. The difference between two adjacent center frequencies is referred as the frequency increment f. A larger f results in fewer scanning steps and higher probability of failing to detect the covert channel, but with a lower runtime cost. Fig. 11 shows the linear scanning method, and the algorithm is listed in **Algorithm 4.** For each core i, an FIR with center frequency f_{center} and bandwith of f is used to filter the thermal signal. If the maximum amplitude of the filtered signal is higher than threshold T_r, it is deemed as a malicious core involved in the thermal covert channel attack. The center frequency of FIR sweeps from f_l to f_h with an incremental of f.

The second scanning method scans the frequency spectrum by randomly selecting the center frequency, in the range of $[f_l, f_h]$, of the FIR filter.

(2) **Recording-based approach**. The above approaches need to run at a fast speed to "track" the thermal covert channel and sometimes might fail to detect an attack. In this approach, the temporal temperature signal is recorded for 1 second, and transformed into frequency domain using fast Fourier transform (FFT) (corresponding to line 2 in **Algorithm 5**). The frequency spectrum of the signal is checked whether there is a potential attack or not. The advantage of this approach is that it can check the full spectrum. However, since recording a

Algorithm 4 The scanning-based approach

Input:
f_{start}: The minimum frequency of detection
f_{end}: The maximum frequency of detection
f: The frequency increment of linear scan
f_{center}: The center frequency of the FIR filter

Output:
f_{detect}: The detected thermal signal frequency
L: The cores that transmit the thermal signal

1: **for** Each core i **do**
2: $f_{center} = f_{start}$
3: **while** f_{center} & f_{end} **do**
4: Filter thermal signal of core i with an FIR with f_{center}, and bandwidth of f
5: **if** The maximum of the filtered signal amplitude $> T_r$ **then**
6: $f_{detect} = f_{center}$
7: Record i as malicious and add i into L
8: **end if**
9: $f_{center} + = f$
10: **end while**
11: **end for**
Return f_{detect}; L

Algorithm 5 The recording-based approach

Input:
f_{start}: The minimum frequency of detection
f_{end}: The maximum frequency of detection

Output:
f_{detect}: The detected thermal signal frequency
L: The cores that transmit the thermal signal

1: **for** Each core i **do**
2: Compute the frequency spectrum of core i's thermal signal
3: **if** The maximum value of the spectrum in $[f_{start}; f_{end}] > T_r$ **then**
4: f_{detect}=The frequency with the maximum value
5: Record i as malicious and add i into L
6: **end if**
7: **end for**
Return f_{detect}; L

long temperature sequence takes time, the attack might already transmit sensitive data before being detected.

Given the output of the FIR filter (approach 1) or the spectrum (approach 2), a decision should be made by comparing the signal amplitude with a given threshold T_r (corresponding to line 5 in **Algorithm 4**, and line 3 in **Algorithm 5**). If the amplitude exceeds the threshold, it can conclude that an attack has been discovered, and the signal frequency and core number (f_{detect} and L), are passed to the jamming process for action.

B. The Jamming Process

Once the thermal covert channel is detected, a random bit sequence of '1's and '0's is generated as noise at **the random noise generation module**. This random sequence $s(t)$ can be expressed as

$$s_{jam}(t) = r_n g_{jam}(t \quad nT_{jam}) \tag{3}$$

where T_{jam} is the period of one bit (symbol width), $g_{jam}(t)$ is the baseband pulse waveform with duration of T_{jam}, and r_n is the value of the n-th bit of the random sequence which is defined as follows:

$$r_n = \begin{cases} 1 & \textit{with a probability of } 0.5 \\ 0 & \textit{with a probability of } 0.5 \end{cases} \tag{4}$$

The single-tone noise generation module, unlike random noise, continuously sends bit sequence of '1's, *i.e.*, $r_n = 1$.

At **the modulation module**, the sequence generated by the noise generation module is modulated by the carrier that has the same frequency as the transmission frequency f_{detect} occupied by the detected covert channel. The output of the modulation module, $e_{jam}(t)$, is thus

$$e_{jam}(t) = s_{jam}(t) \quad c(t) \tag{5}$$

where $s_{jam}(t)$ is the noise generated by the noise generation module, and $c(t)$ is the carrier with a frequency of f_{detect}.

Finally, the thermal signal is generated following the same temperature signal generation approach as described in Section III-C.

C. Time Overhead

For the thermal covert channel that can change its frequency, the jammer will need to track the frequency of the thermal covert channel and learn all the frequencies used by the thermal covert channel. Denote T_a as the time for the detection unit to find a thermal covert channel signal, and T_b as the time for the transmitter to find itself is being jammed. After T_b, the transmitter starts to change its communication frequency, and the time required for the transmitter to successfully change the frequency is T_c. For the frequency list with M frequencies, the time overhead for the jammer to complete the learning of its entire frequency list is $(T_a + T_b + T_c) \quad M$. A longer detection time increases the probability that the attacker can leak more information before the countermeasure unit detects all the possible frequencies and blocks them.

V. Experimental Evaluation

A. Experimental Setup

Our experiments are performed on two different platforms. The first is a many-core simulator with an integrated power model [23], and hereinafter, this platform is referred as the simulation platform. We simulate both 2D and 3D many-core chips with different levels of power consumption pertaining to the processor cores. All the parameter values for different configurations are listed in Table I. The floorplan of the processor cores follows the one reported in [24]. In these

TABLE I
THE CONFIGURATIONS FOR THE SIMULATION.

Core Architecture	Alpha 21264
Number of Cores	3 3 3
CPU Frequency	2000MHz
Fetch/Decode/Commit size	4 / 4 / 4
L1 D cache	16KB, 2-way, 32B line, 2 cycles, 2 ports
L1 I cache	32KB, 2-way, 64B line, 2 cycles
L2 cache	64KB, 64B line, 6 cycles, 2 ports
Main memory size	2GB
Chip thickness	0.00015m
Silicon thermal conductivity	$100W/(m \quad K)$
Silicon specific heat	$1{:}75 \quad 10^6 J/(m^3 \quad K)$
Temperature threshold for DTM	$373{:}15K$
Heat sink side	$0{:}06m$
Heat sink thickness	$0{:}0069m$
Heat sink thermal conductivity	$400W/(m \quad K)$
Transmission frequency list	[40Hz,50Hz,60Hz,70Hz,80Hz,90Hz]

TABLE II
THE CONFIGURATIONS FOR THE EXPERIMENT ON A REAL MACHINE.

Processor	Intel i7-6700k @4.0GHz
Memory	16 Gbytes
DRAM Frequency	1200MHz
Mainboard	MSI Z170-A PRO
Physical Cores	4
Logical Cores	8
Fan Speed	1200rpm
Operate System	Ubuntu 16.04.5 LTS
Dynamic Fan Speed	OFF

experiments, a transmitter is set to be 1 hop away from the receiver, and both transmitter and receiver are vertically adjacent to each other. The chip temperature is simulated using HotSpot [14], an accurate temperature simulator. The power trace of each core is fed as the input into the HotSpot to get the temperature.

The second sets of experiments are performed in a real computer, whose configurations are listed in Table II. The CPU is Intel i7-6700k @4.0GHz with the operation systems of Ubuntu 16.04.5 LTS. We run the **modprobe msr** instruction to load the msr module for the temperature sensor reading and set the CPU frequency governor to the performance mode with the **cpufreq-set –g performance** instruction. In this case, the CPU frequency is fixed to be 4.0GHz. In these experiments, two hardware threads in a core are dedicated to operate as a thermal covert channel, while other cores are loaded with various applications, including user applications (*e.g.*, browsing the web), AES [25], and those from PARSEC [26]. The AES is one of the computationally intensive applications. All these applications running at different cores pose as noise to the thermal covert channel.

In the experiment we test the performance of a thermal covert channels based on the measure of PER (packet error rate), defined as

$$PER = \frac{N_e}{N} \quad 100\% \tag{6}$$

where N is the total number of packets transmitted, N_e is the number of erroneous packets (those failed the cyclic redundancy check) and lost packets combined.

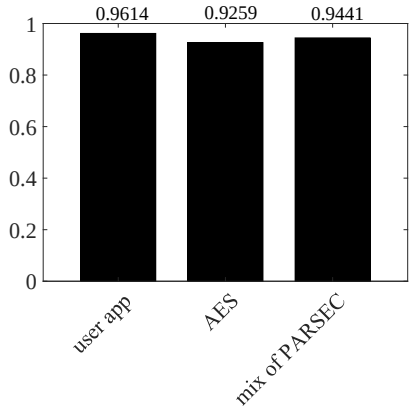

Fig. 12. The R-squares for different noises in Gaussian fitting.

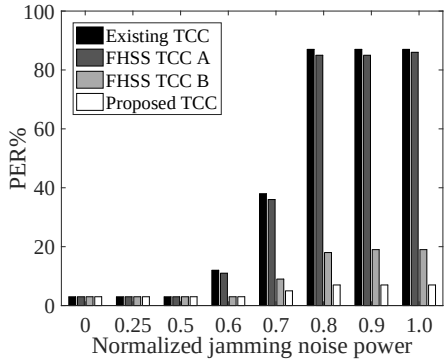

Fig. 13. The PERs by varying the jamming noise power.

B. Evaluating the Enhanced Attack and Countermeasure

1) Evaluating the Channel Noise: In the first set of experiments, the thermal noise is evaluated by running three applications: the user applications, the AES and the PARSEC. Here the thermal noise is treated as white noise with zero mean and finite variance. The experiment is performed on the real machine. The temperature is recorded every 10 ms under three different system loads (treated as noises sources).

We use the following Gaussian function to fit the probability distribution function (pdf) of the noises,

$$f(x) = ae^{-(\frac{x-b}{c})^2} \tag{7}$$

where b is the expectation, c is the standard deviation, and a is the height of function curve peak. Fig. 12 shows the R-squares of the pdf of the noises. The closer the R-square to 1, the closer the system noise to Gaussian noise.

As shown in Fig. 12 the R-squares are 0.9614, 0.9259 and 0.9441 for the user applications, the AES, and the PARSEC, respectively. As all R-square values are reasonably close to 1, it is safe to treat the noise follows the Gaussian distribution.

2) Evaluating the Proposed Thermal Covert Channel: Fixed-frequency jamming noise is inserted in this set of experiments. Fig. 13 compares the PERs of our dynamic frequency changing thermal covert channel with those of a conventional FHSS using an independent channel (100 Hz) for synchronization (herein referred as FHSS TCC A), another conventional FHSS using the same communication channel

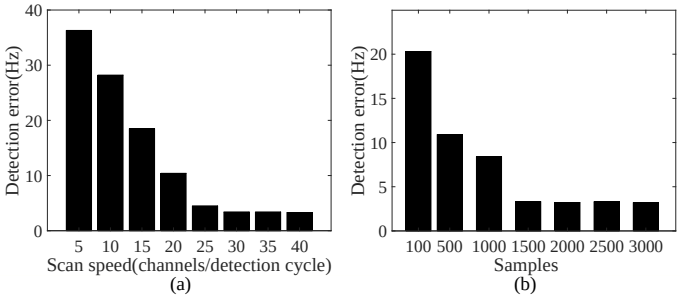

Fig. 14. (a) Detection errors of the scanning-based approach with varying scan speeds. (b) Detection errors of the recording-based approach with varying sample numbers.

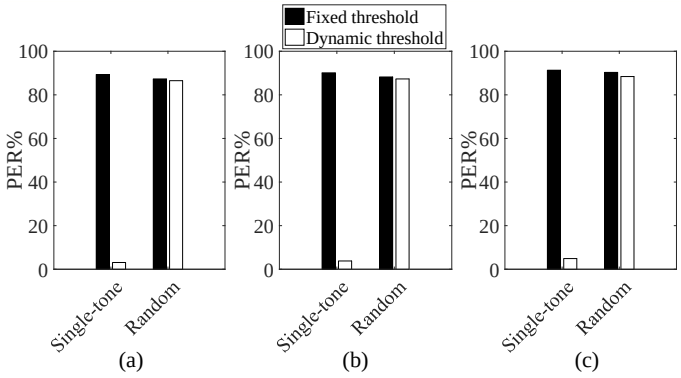

Fig. 15. (a) PER comparison by varying the threshold and jamming method in the 0-hop simulation. (b) PER comparison by varying the threshold and jamming method in the 1-hop simulation. (c) PER comparison by varying the threshold and jamming method in the real machine.

for synchronization (herein referred as FHSS TCC B) [27], and the existing thermal covert channel in [3] with different jamming noise powers. The normalized jamming noise power is defined as the ratio of the jammer's jamming power to the thermal covert channel's signal power. The power overhead of a thermal covert channel is $23.43W$. One can see from Fig. 13 that the conventional FHSS TCC A has a very high PER, because when an independent channel used for synchronization is jammed, no communication can occur between the transmitter and the receiver. FHSS TCC B uses the thermal covert channel also for synchronization purpose; the jammer with a fixed jamming frequency blocks the transmission of the synchronization information when the jamming frequency matches the communication channel frequency. Therefore, FHSS TCC B has a higher PER than the proposed TCC. In addition, its frequency hopping synchronization information consumes extra communication bandwidth. One can see from Fig. 13 that when the jamming noise power equals to the thermal covert channel signal power, the PER of the existing thermal covert channel [3] is over 85%, while the PER of the proposed covert channel is 5%. The reason is that the jammer detects the thermal covert channel and emits the noise signal with the same transmission frequency as the thermal covert channel. The receiver of the existing thermal covert channel [3] reads both the noise and the packets from the transmitter, which results in decoding errors. However, our dynamic frequency changing protocol can detect the jamming and changes the transmission frequency dynamically to avoid being jammed. Once the transmission frequency moves to a new frequency, the fixed-frequency jammer loses its target to jam. This is the main reason that the PER of the proposed thermal covert channel is lower than the existing thermal covert channel.

3) Evaluating the Detection Scheme: Fig. 14 compares the detection error of the two detection methods described in Section IV-A under different setups and parameters.

The scanning-based approach is performed over the entire frequency band from 30Hz to 150Hz. Scan speed is defined as the number of channels processed for a complete scan of the entire frequency band. Scanning through the entire frequency band completes one detection cycle. Detection error is defined as the absolute value of the difference between the true transmission frequency and the detected frequency of the thermal covert channel. Fig. 14(a) shows that the detection error decreases with the increase in the scan speed. When the speed is greater than 30 per detection cycle, the detection error is as small as 3Hz.

In the recording-based approach, we use different numbers of samples (defined as the number of data points collected from the sensor to perform FFT). Fig. 14(b) shows that more samples lead to smaller detection errors. When the number of samples is greater than 1500, the detection error is 3Hz. In the following experiments, the recording approach will be used due to its lower overhead.

4) Evaluating the Countermeasure Schemes: The effects of the two proposed countermeasure approaches against the enhanced covert channel attack in Section III with the two decision thresholds (in Section III-A) are evaluated in Fig. 15. When a fixed decision threshold is used in the covert channel, both single-tone and random noises jam the enhanced covert channel effectively. In the 0-hop (the transmitter and the receiver are two hardware threads in a real machine multi-threading processor) and 1-hop (the transmitter and receiver are two adjacent cores in vertical direction) experiments, the PERs of the proposed thermal covert channel increase to over 85%. The PER of the proposed thermal covert channel in the real machine even reaches 90%, literally making any information leaking impossible. One can see from Fig. 15 that the PER under the single-tone noise jamming is higher than that of the random noise against the covert channel using a fixed threshold. When a dynamic decision threshold is used in the covert channel, the single-tone noise jamming cannot block it. The thermal covert channel still maintains a very low PER, less than 5%. The reason is as follows. From Equations (1) and (2), the modulated covert channel signal can be expressed as

$$signal(t) = a_n g(t \quad nT_B) \quad c(t) \tag{8}$$

From Equations (3) and (5), the modulated single-tone noise jamming signal can be expressed as

$$noise(t) = g_{jam}(t \quad nT_{jam}) \quad c(t) \tag{9}$$

To simplify analysis, we assume that the jamming noise has the same phase, symbol width (T_B, T_{jam}), and baseband pulse waveform ($g(t)$, $g_{jam}(t)$) as the covert channel signal. The signal received by the receiver is the superposition of the jamming noise and the covert channel signal, which is

$$receive(t) = \begin{cases} 2g(t - nT_B) & c(t) + n(t) \quad a_i = 1 \\ g(t - nT_B) & c(t) + n(t) \quad a_i = 0 \end{cases} \tag{10}$$

where $c(t)$ is the carrier signal, and $n(t)$ is the white noise generated by the system. If the amplitude of the transmission signal, the jamming signal, and the white noise are , , and , respectively, with the single-tone noise jamming, the amplitude of the received signal is + + when the transmitter sends a '1', and + when a '0' is transmitted. The dynamic decision threshold in the covert channel is set to be (+ +)=2. If > , (+ +)=2 > (+), *i.e.*, the receiver of the covert channel shall make the correct decision. Therefore, the single-tone noise jamming cannot block the covert channel when it uses the dynamic decision threshold. However, when a fixed decision threshold of is adopted in deciding the received signal symbols from the thermal covert channel, the symbols would be wrongfully taken as logic '1' (as the amplitude at the receiver is over the threshold), resulting in a high PER for the single-tone noise jamming.

In contrast, the random noise jamming still can effectively block the thermal covert channel, whose PER rises to over 85% in both simulations and the real machine.

Table III further compares the PERs of FHSS TCC A, FHSS TCC B, and the proposed TCC when our proposed countermeasure is applied. One can see that our proposed countermeasure can effectively block information leak caused by FHSS TCC A, FHSS TCC B, and our proposed TCC with very high PERs.

C. Runtime and Power Overheads of the Countermeasure

The scanning-based approach with a scan speed of 30 samples per detection cycle takes a total of 50 milliseconds to complete a scan of the entire frequency band to detect a covert channel. The recording-based approach needs to compute 1500-point FFT, which takes less than 8 milliseconds. When the frequency of possible attacks is detected, the defender generates jamming noise with a duration of up to tens of seconds. Therefore, the total time of running the countermeasure takes less than 50 milliseconds. For the recording-based approach, the time overhead is even less than 10 milliseconds and the average power consumption due to jamming is $22.27W$.

For the thermal covert channels whose frequency can be changed, from the simulation, one can see T_a is 50 milliseconds, T_b is 10 milliseconds, and T_c is 2 seconds in the best scenario (the receiver can receive DFCQ packets directly) and 6.8 seconds in the worst (using polling) scenario (see Section III-B). As the length of the frequency list in the experiment is 6, the jammer can span the entire frequency list within 12.36 and 41.16 seconds in the best and worst scenarios, respectively.

VI. Conclusion

It was shown in the paper that a thermal covert channel could acquire strong anti-jamming capability by following a communication protocol that enables the thermal covert channel to reliably change its communication frequency. In order to combat this enhanced covert channel, a countermeasure was proposed, which detects an attack by scrutinizing the frequency spectrum periodically. It then reactively emits a jamming noise with the same frequency of the covert channel. Experimental results confirmed that the PER of the enhanced covert channel reduces to as low as 5%. However, when the proposed countermeasure is applied, its PER jumps to 85%, effectively shut down thermal covert channel attacks even with enhanced capabilities.

TABLE III
THE PERs OF FHSS TCC A,FHSS TCC B, AND THE PROPOSED TCC WITH THE PROPOSED COUNTERMEASURE APPLIED.

	FHSS TCC A	FHSS TCC B	Proposed TCC
PER	90.2%	87.8%	87.6%

Acknowledgment

This research program is supported by the Natural Science Foundation of Guangdong Province No. 2018A030313 166, Pearl River S&T Nova Program of Guangzhou No. 201806010038, the Fundamental Research Funds for the Central Universities No. 2019MS087, Open Research Grant of State Key Laboratory of Computer Architecture Institute of Computing Technology Chinese Academy of Sciences No. CARCH201916, Key Laboratory of Big Data and Intelligent Robot (South China University of Technology), Ministry of Education, and the Natural Science Foundation of China No. 61971200.

References

[1] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S. Capkun, "Thermal covert channels on multi-core platforms," in *Usenix Conference on Security Symposium*, 2015, pp. 865–880.

[2] D. B. Bartolini, P. Miedl, and L. Thiele, "On the capacity of thermal covert channels in multicores," in *European Conference on Computer Systems*, 2016, p. 24.

[3] Z. Long, X. Wang, Y. Jiang, G. Cui, L. Zhang, and T. Mak, "Improving the efficiency of thermal covert channels in multi-/many-core systems," in *Design, Automation & Test in Europe Conference & Exhibition*, 2018, pp. 1459–1464.

[4] M. Guri, M. Monitz, and Y. Elovici, "Usbee: air-gap covert-channel via electromagnetic emission from usb," in *Conference on Privacy, Security and Trust*, 2016, pp. 264–268.

[5] X. Zhang, Y.-A. Tan, C. Liang, Y. Li, and J. Li, "A covert channel over volte via adjusting silence periods," *IEEE Access*, vol. 6, pp. 9292–9302, 2018.

[6] M. Guri, O. Hasson, G. Kedma, and Y. Elovici, "An optical covert-channel to leak data through an air-gap," in *Conference on Privacy, Security and Trust*, 2016, pp. 642–649.

[7] N. Matyunin, J. Szefer, S. Biedermann, and S. Katzenbeisser, "Covert channels using mobile device's magnetic field sensors," in *Asia and South Pacific Design Automation Conference*, 2016, pp. 525–532.

[8] L. Deshotels, "Inaudible sound as a covert channel in mobile devices," in *Usenix Conference on Offensive Technologies*, 2014.

[9] E. Tromer, D. A. Osvik, and A. Shamir, "Efficient cache attacks on AES, and countermeasures," *Journal of Cryptology*, vol. 23, no. 1, pp. 37–71, 2010.

[10] S. Cabuk, C. E. Brodley, and C. Shields, "IP covert timing channels: design and detection," in *Conference on Computer and communications security*, 2004, pp. 178–187.

[11] J. Chen and G. Venkataramani, "Cc-hunter: uncovering covert timing channels on shared processor hardware," in *International Symposium on Microarchitecture*, 2014, pp. 216–228.