SSPS: An UEFI Based Secure System-in-Pocket-Storage Approach to Desktop-Go-With-Person

Ming Wu
Intel Corporation
Shanghai, China
ming.m.wu@intel.com

Lei Zhou
Southern University of Science and Technology
Shenzhen, China
zhoul6@sustech.edu.cn

Vincent Zimmer
Intel Corporation
Oregon, US
vincent.zimmer@intel.com

Michael Rothman
Intel Corporation
Oregon, US
michael.a.rothman@intel.com

Fujin Huang
Intel Corporation
Shanghai, China
fujin.huang@intel.com

***Abstract*—The invention of personal computer together with its rapid deployment around the world are among the most important achievements that have changed lives of people in this planet. In modern society, people are becoming more dependent on computer devices. However, it's difficult to bring computer hardware with us at any time and in any place, which is known as the desktop-go-with-person (DGWP) problem. Regarding it's easy to find a nearby computer and bring tiny non-volatile memory devices with us, putting the entire desktop system software into pocket storage and loading it on any available hardware become an attractive approach. However, current computer software are tightly bound to hardware. Live USB systems are quite popular for trial run purpose; however, they are read-only system and limited to certain OSes. Also, the creating of live USB system from native installed desktop system is dependent to OS. In this paper, we proposed a new generic UEFI-based secure system-in-pocket-storage approach (SSPS) to address the DGWP problem by decoupling native installed system software from hardware, making software go with person and running it with any other computer hardware. The paper illustrated the SSPS overall block diagram and workflow, internal architecture, data structure and management of desktop system image, image loader and virtual disk driver, and security considerations during the design and implementation. Also, the paper provided system evaluation of SSPS approach as well as technology forecasts.**

Keywords—Secure System-in-Pocket Storage (SSPS), Non-volatile Memory (NVM), Unified Extensible Firmware Interface (UEFI), Desktop-go-with-Person (DGWP)

I. Introduction

The invention of personal computer and its rapid and mass deployment around the world are one of the most important achievements during the past decades that has changed human lives in this planet. In modern society, people are becoming dependent on computer devices - work, study, shopping, entertainment and social networking. People prefer to use personal computer for security and user experience considerations. However, it's impossible or not convenient to bring computer at any time like travelling or gatherings with friends. It is known as desktop-go-with-person (DGWP) problem.

There are existing approaches to address the DGWP problem, such as making computer smaller and lighter [1][2], cloudifying legacy applications [3] for accessing from handheld devices, using live USB system [6][7][8], and network based desktop migration[9-13]. However, they have such limitations as big formfactor or inadequate computing capability, the cloudification effort of legacy software, and slow responsiveness of user interaction caused by network latency.

Fortunately, it's not difficult to find computers in a nearby iCafe, library, or borrow from friends. During the past decades, non-volatile memory (NVM) technologies [14][15] have been developing rapidly. Various NVM pocket storage devices have come out like inexpensive USB disks or thumbs. The physical size of these devices is small enough to bring anywhere. Inspired by the idea of Transparent Computing [16][17] to decouple software from hardware, we are thinking of splitting software from hardware platform, storing the entire software and data to pocket storage, leveraging available hardware to run it, and recording the changes in guest hardware and sending back to original platform. The pocket storage could solve the desktop-go-with-person (DGWP) problem. However, the entire software stack, from OS, middleware to applications, is tightly bound to hardware. Privacy and data security are also barriers. The people who owns the hardware may hesitate to lend device to others, while people who borrow devices are also worried about leaving private data in borrowed computers. Although it looks alike, live USB system only runs in read-only mode and cannot keep personal setting for next boot. It only works for special OSes like Ubuntu and Windows PE by specific tools. What's more, it lacks security measures to protect data and system in USB thumb and guest computers.

In this paper, we proposed a secure system-in-pocket-storage approach (SSPS) to address the DGWP problem. SSPS is based on the next generation Unified Extensible Firmware Interface (UEFI) technology [18] as well as author's work on transparent client infrastructure (TCI) [19-21] at Intel. SSPS solves DGWP problem by 1) making software go with person as image files in pocket storage other than moving entire computer device; 2) loading system image with a BIOS level SSPS loader and running it on native bare metal, not VM; 3) the design of system image management in pocket storage and making it work with SSPS virtual disk driver on different hardware natively; 4) the mechanism of storage updating during run-time period and moving it back to original platforms; 5) security considerations to protect data and system in both pocket storage and host platforms.

The contribution of this paper includes the proposal of SSPS approach to decouple system software from hardware and make it go with person through pocket storage, the organization and management of desktop system image, the design of SSPS loader and virtual disk driver, the working prototype, security mechanism, a list of performance indicators to analysis and evaluate the approach as well as the forecast of future directions. The rest of paper is organized as follows. Chapter 2 introduces the SSPS background and concept, and why it's needed. Chapter 3 lists existing technologies and the limitations. Chapter 4 gives SPSS overview. Chapter 5 presents SPSS design in detail. Chapter 6 give security considerations. Chapter 7 and 8 reports the implementation and evaluation. In chapter 9, challenges and future directions are provided. The final chapters are conclusion and acknowledgement.

II. Background and Concept of SSPS

Before illustrating the Secure System-in-Pocket Storage (SSPS) principle and how it works, the following concepts need to be clarified. Figure 1 shows the relationship of SSPS building blocks.

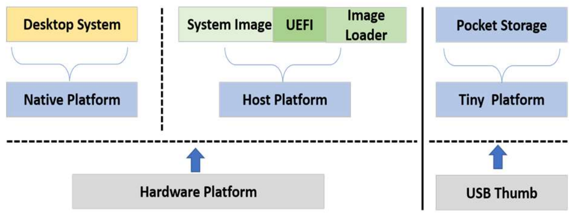


Fig. 1. Concept of SSPS Building Blocks

Native and host platform: the original hardware that entire desktop system software runs, and the temporary hardware that software image in pocket storage runs.

Desktop system and desktop system image: the set of system software files that runs on native platforms, and image files to store desktop system. System software includes OS, middle-ware, and applications.

Base and delta image: the initial system image file which is created in native platform, and the incremental image files to the base image which are created during the execution of desktop system in host platforms.

UEFI BIOS and image loader: UEFI is the next generation platform BIOS specification that defines a new model for the interface between personal-computer operating systems and platform firmware. UEFI significantly improves the modularity of platform firmware and enable us add modules during pre-boot period. Image loader is an UEFI module to load and run desktop system image in the SPSS design.

Pocket storage and USB thumb: pocket storage is a tiny, portable mass storage device to go with person. USB thumb is a kind of non-volatile memory or flash-based pocket storge to store desktop system image.

The initiative of SSPS approach is to put desktop software system into pocket storage, bring it with person, load and run it natively on any hardware. This technology brings a new mechanism for telecommuting, so people do not need to bring laptop anymore. It also boosts such new business as computer renting by plugging the pocket storage to a rented hardware, or new IT administration mechanism by software system refresh to new hardware via pocket storage.

III. Limitation of Existing Technologies

Some existing technologies have been proposed to address the DGWP problem. The size and weight reduction of computer device from laptop [1] to recent computer stick [2] are the continuous attempts. However, it's still too big in size or lack of computing capability. Cloudification of legacy applications [3] and having them accessed via handhold devices are alternative approaches, however it's not suitable for large software like CAD or video processing with small screen devices. Thin clients and personal cloud desktops have big screen to access remote VMs via ICA or RDP [4][5], but the responsiveness of user interaction caused by network bandwidth and latency is a problem. What's more, the cloudification of legacy software are not always viable due to either technical or business reasons.

Live USB system like Rufus [6] for Ubuntu is another approach. S. Potter et al proposed a DeskPod design [7] that suspended desktop system to portable storage and resumed vice versa. However, they only work on Linux. Benjamin et al proposed a pocket ISR solution [8] by running a hypervisor on live USB and download VM images from cloud. But apparently, virtual machine will have performance discount.

Network-based desktop system migration are viable approaches for DGWP by migrating process, VM or container. Steven et al proposed a process group migration solution [9] with a thin virtualization layer at OS level. Jacob et al proposed an on-the-fly OS migration solution [10] with a Xen-based hypervisor. Haikun et al provided a CR & TR (checkpointing/recovery and trace/replay) mechanism [11] for fast and transparent VM migration. Shripad et al designed a just-in-time live container migration [12] with file-system agnostic service. Bo, Song et al proposed an efficient live migration system Sledge [13] with layered image and selective management context migration. However, these solutions are dependent on network connections, which are not always available and reliable.

IV. Overview of SSPS

A. Problem Statement

The objective of SSPS is to find a generic approach to split natively installed software from hardware, put the software into pocket storage and run it with available hardware. It is broken down into the following problems: 1) how to decouple desktop software system from native platform and put it into pocket storage; 2) how to load the system image at host platform; 3) how to run the system image from pocket storage with bare metal and record the updates; 4) how to move and merge the pocket storage back to the original platform; 5) how to make data and system secure during the work flow; 6) how the approach could be generic for all mainstream PC platforms and operating systems.

B. Scope and Prerequisite

The scope of SSPS technology is: 1) it's for PC hardware, not handhold or embedded platforms. 2) it supports mainstream OSes including Win7, Win10, and various Linux distributions. 3) it regards the entire software as a blob and does not distinguish between OS, middleware, applications and data files. 4) the original native platform and host platform could be different in hardware configuration. 5) security and data privacy should be carefully taken into consideration. 6) the performance is expected to be comparable to native.

C. *Brief*

The SSPS approach is to decouple desktop system software from native platform, load and run it on any available host platforms via pocket storage devices. The characteristics of SSPS include: 1) the software in pocket storage could run on both bare metal for performance, or virtual hardware for security and compatible considerations. 2) it's BIOS-based software solution, so without hardware cost. 3) it's a generic technology for all mainstream desktop OS including close-source windows and open-source Linux. 4) the incremental based system image management mechanism for flexible tracking, updating, and merging of virtual storage.

D. *Key Indicators*

A list of indicators is shown in table 1 to measure the SSPS design. It includes the boot and run-time performance of system image in host platform compare with that in native platform, as well as the migration performance from native platform to pocket storage and vice versa.

TABLE 1 KEY INDICATORS

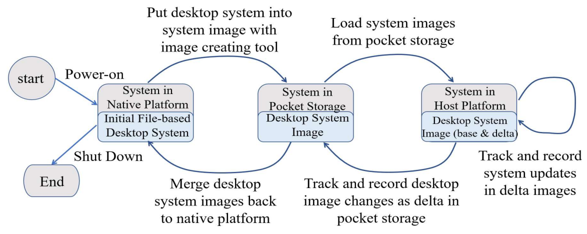
Indicator category	Description	Key indicators	Remark
Boot overhead	System OS initialization	Mainstream OSes	vs. legacy boot
Run-time overhead	System run-time effectiveness	Application loading, running and I/O throughput	Compare with native system.
Migration overhead	Pocket storage storing and restoring	Memory and storage fetching and rewriting	Consider the image size and I/O throughput.

V. DESIGN OF SSPS

The SSPS approach addresses the DGWP problem by splitting the desktop software system from native platform to pocket storage as desktop system images and running the images at host platforms from pocket storage. In this section, a deep-dive analysis of SSPS approach will be provided, including the overall block diagram and workflow, SSPS host and pocket storage architecture, data structure and management of desktop system image, virtual disk driver internal architecture, and how OS runs on system image.

A. *Block Diagram and Workflow*

Figure 2 shows the SSPS block diagram and workflow.



Put desktop system into system image with image creating tool
Load system images from pocket storage
start
Power-on
System in Native Platform
Initial File-based Desktop System
System in Pocket Storage
Desktop System Image
System in Host Platform
Desktop System Image (base & delta)
Shut Down
End
Merge desktop system images back to native platform
Track and record desktop image changes as delta in pocket storage
Track and record system updates in delta images

Fig. 2. SSPS Approach Workflow

The system image was created from desktop system in native platform by image creating tool and transferred to pocket storage. The pocket storage was brought and attached to remote host platform where it was loaded. During the run-time, all block IO updates were not written back to the original system image, but to the delta images in pocket storage with copy-on-write mechanism. When host platform was shut down, the pocket storage would be brought to another host, attached there and a second delta image over previous one was created. The above steps was repeated for multiple times until finally the pocket storage was sent back to the original native platform again, with initial system image and multiple delta images. The image merging tool merged these images back to native platform; therefore, the latest desktop system was transferred back.

In summary, the pocket storage might be regarded as the intermediate media between one native platform and multiple host platforms to run the end user's personal desktop system one by one. Tools in USB thumb – the image creating, loading and merging tool – were used to take native desktop software system out of native platform to pocket storage and sent back.

B. *SSPS Host and Pocket Storage Architecture*

Compare with native PC's OS loading from local hard disk as data blocks, SSPS host must load system from system images in pocket storage. SSPS host has a different internal architecture from native PC as in left of figure 3.

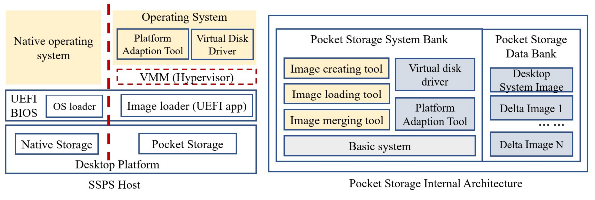


Fig. 3. Architecture of SSPS Host and Pocket Storage

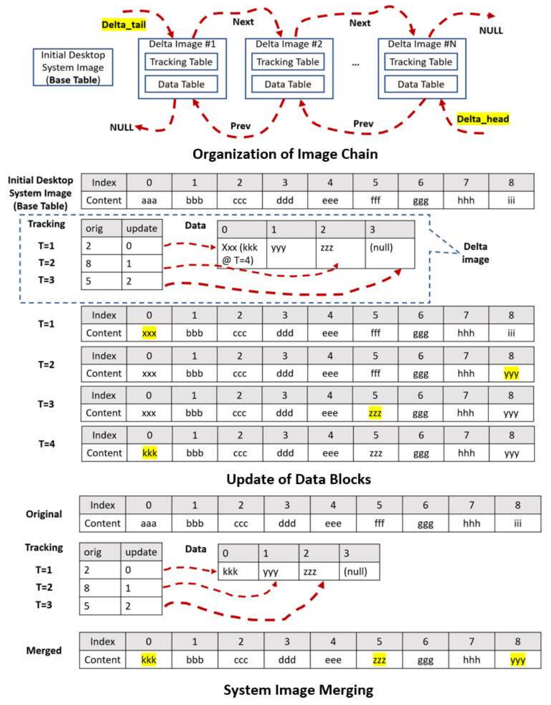
When system boots, the SSPS host BIOS detects whether the pocket storage is attached. If not, SSPS host continues to boot OS from local storage as normal PC, otherwise it loads system image from pocket storage with SSPS image loader. SSPS image loader is a special UEFI application within UEFI BIOS to check and load the desktop system image from pocket storage during pre-boot period. During the OS run-time, the virtual disk driver would redirect OS level IO read & write to system images in pocket storage.

Logically, there are two banks in pocket storage: 1) system bank, where basic system is loaded by BIOS to set up the underlying running environment for three tools below. It could be UEFI or embedded OS environment. The three tools are in system bank: A) image creating tools to create the system image from native platform. B) image loading tools to load the system image file in host platform. C) image merging tools to merge system image and delta images and move back to native platform. All three tools are running on the basic system. 2) data bank, which is the other partition to store desktop system images created by image creating tools and delta images created during run-time. Figure 3 right is the logical layout of pocket storage system.

When the system image was created, the virtual disk driver and platform adaption tool were injected after the dumping of desktop system content. The platform adaption tool was to detect host platform type, enumerated device drivers in database and installed suitable driver modules accordingly. In addition, a matching mechanism before loading the system image was added in image loader to determine whether a hypervisor or virtual machine manager (VMM) was needed. A meta data region was created in each desktop system image to indicate the minimal hardware requirement, and used to compared with SSPS host's real hardware, such as processor type, memory, and the list of required peripherals like graphics, together with security expectations. If it meets, the system image will be loaded on local bare metal, otherwise a hypervisor will be started to simulate the required platforms. What's more, the platform adaption tools may help on installing hardware drivers for bare metal adaption.

Although, the system image could work on both native bare metal hardware and virtualized platform, bare metal is always preferred for better performance and user experience. The hypervisor will be chosen only when, 1) the platform is too new to run legacy OS, such as Win7 on latest Intel Alder Lake platform. 2) security consideration is a must, as the host wants to completely isolate guest desktop system from hardware with hypervisor layer. If host platform was incapable of loading system images either natively or on hypervisor, it would give a warning and quit.

C. *Incremental Based Desktop System Image*



Index	0	1	2	3	4	5	6	7	8
Content	aaa	bbb	ccc	ddd	eee	fff	ggg	hhh	iii

	orig	update
T=1	2	0
T=2	8	1
T=3	5	2

0	1	2	3
Xxx (kkk @ T=4)	yyy	zzz	(null)

	Index	0	1	2	3	4	5	6	7	8
T=1	Content	xxx	bbb	ccc	ddd	eee	fff	ggg	hhh	iii
T=2	Content	xxx	bbb	ccc	ddd	eee	fff	ggg	hhh	yyy
T=3	Content	xxx	bbb	ccc	ddd	eee	zzz	ggg	hhh	yyy
T=4	Content	kkk	bbb	ccc	ddd	eee	zzz	ggg	hhh	yyy

	orig	update
T=1	2	0
T=2	8	1
T=3	5	2

0	1	2	3
kkk	yyy	zzz	(null)

	Index	0	1	2	3	4	5	6	7	8
Original	Content	aaa	bbb	ccc	ddd	eee	fff	ggg	hhh	iii
Merged	Content	kkk	bbb	ccc	ddd	eee	zzz	ggg	hhh	yyy

Fig. 4. Image Chain in Pocket Storage System and Organization of Desktop System Image

In this section, how system images are organized and managed in pocket storage will be explained. In pocket storage, there are multiple system image files, including an initial system image and multiple delta images. The delta images are used to record the changes to previous image. The image loader in pre-boot period and virtual disk driver in run-time period can get any latest data blocks from image files.

Internally, the delta image includes a tracking table and a data table to track the offset and content of changed block respectively. Each incremental updates to the initial system image are kept tracked in delta image, so it's easy to roll-back and merge into the original native platform. The top of figure 4 shows how the image chain is organized.

Let's give an example on how data blocks are updated as middle of figure 4. The OS storage is split to equal size data blocks like block #0, block #1, and so forth. During run-time period, any software activities, including software upgrading, system patching, file writing, or system settings, will lead to the data block updates in hard disks. A virtual disk driver is plugged into OS kernel to capture disk IOs. When IO writes are detected, it will be recorded in latest delta image. In time 1, block 2 "ccc" was updated to "xxx", a record was added to tracking table as well as a data block in data table. Same was the 8th block "iii" to "yyy" and 5th block "fff" to "zzz" during time 2 and 3. In time 4, block #0 was replaced by "kkk", the first items of tracking table and data table was modified accordingly. The snapshot of system image content from time 1 to 4 are shown. The bottom of figure 4 shows how system image is merged back to native platform. The delta image chain is walked through from the earliest to latest, and the tracking table in each delta image will be written back accordingly to the initial system image.

Algorithm 1: ***Block_Read****(base, delta_head, idx)*

Input: Base is data blocks of initial desktop system image, and delta head is head (or latest) of the delta image chain. Each delta image includes tracking table (tracking) and data table (data). Idx is the index of block IO to be read
Output: the content of idx-th block IO

```
 1: Begin Out:
 2: Walk through delta image chain headed by "delta head", pick up each
delta image in variable "delta"
 3: Begin In:
 4: // check if any tracking table records the change of block idx#
 5: te = delta -> tracking.find(idx)
 6: if te ! = NULL then
 7:      return delta -> data[te.update]
 8:      delta = delta -> prev
 9: end if
10: // Otherwise, return the original block in base
11: End In
12: return base[idx]
13: End Out
```

The algorithms of block IO read (algorithm1) and write (algorithm 2), and the merge of delta images back to initial base image (algorithm 3) are shown here.

Algorithm 2: ***Block_Write****(base, delta_head, idx, content)*

Input: **Base**, **delta_head and Idx** are same as algorithm 1, **content** is data block content to be written
Output: success or failure of the writing operation.

```
 1: Begin:
 2: if delta head == NULL: then // new desktop system image
 3:      delta head = allocate ()
 4:      delta head!prev = NULL
 5:      delta head!next = NULL
 6: end if
 7: delta = delta head // just record to the latest delta image
 8: tracking = delta.tracking
 9: data = delta.data
10: te = tracking.find(idx)
11: if te == NULL then
12:      te = tracking.create() // create an entry in tracking table
13:      te.orig = idx
14:      data[data.size] = content
15:      te.update = data.size++
16: else
17:      data[te.update] = content
18: end if
19: return Success
20: End
```

D. *Read and Write with Virtual Disk Driver*

The previous section illustrates how system images are managed in pocket storage. However, during the implementation, there are two obvious issues in data block reading and writing from pocket storages: 1) IO performance of USB thumb is slower than internal storage like SATA or

NVME disk [22]. 2) reliability of pocket storages is a problem [23]. To cope with the issues, SSPS virtual disk driver is designed as the figure 5 architecture.

The virtual disk driver is placed between user processes and pocket storage devices for read and write of data blocks. Virtual storage interface is the interface to user processes. Memory buffers, including base image buffer, in-memory data block table and tracking table, are located between virtual storage interface and physical pocket storage as IO cache. Regarding the pocket storage is not as reliable as internal storages, a certain number of unused raw storage space from local hard disk is allocated as pocket storage backups.

Algorithm 3: Image_*Merge* *(base, delta_tail)*

Input: **Base** is same as algorithm 1, **delta_tail** is tail (or earliest) of delta image chain.
Output: successful or failure of this merge

```
 1: Begin:
 2: Delta = delta tail
 3: while delta != NULL do // traverse the delta image chain
 4:     tracking = delta->tracking
 5:     data = delta->data
 6:     I = 0
 7:     while I < tracking.size do
 8:         te = tracking[i]
 9:         Base[te.orig] = data[te.update]
10:         i = i+1
11:     end while
12:     Delta = delta!next
13: end while
14: return Success
15: End
```

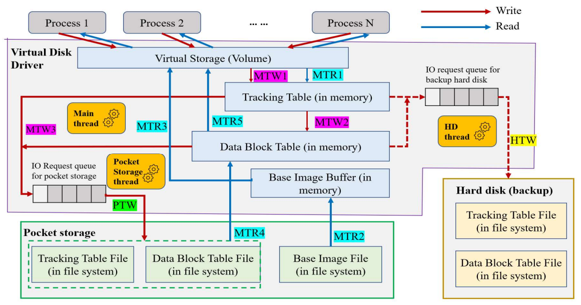

Process 1
Process 2
... ...
Process N
Write
Read
Virtual Disk Driver
Virtual Storage (Volume)
MTW1
MTR1
IO request queue for backup hard disk
Tracking Table (in memory)
Main thread
MTW3
MTR3
MTR5
MTW2
HD thread
HTW
Data Block Table (in memory)
IO Request queue for pocket storage
Pocket Storage thread
Base Image Buffer (in memory)
PTW
Pocket storage
MTR4
MTR2
Tracking Table File (in file system)
Data Block Table File (in file system)
Base Image File (in file system)
Hard disk (backup)
Tracking Table File (in file system)
Data Block Table File (in file system)

Fig. 5. Virtual Disk Driver Architecture

There are three kernel threads in virtual disk driver: 1) main thread. It receives read & write requests from applications and completes it via in-memory base image buffer, tracking table and data block tables. 2) pocket storage thread that receives write request from main thread through IO request queue and writes data blocks to pocket storage. 3) HD thread that receives write requests from main thread and write data blocks to hard disk as pocket storage backup.

When a read request comes through virtual storage interface as in figure 5, it will first check tracking table (MTR1, main thread read #1). If it's not there, it either reads from base image buffer (MTR3) directly or issues a read request from physical pocket storage (MTR2) to base image buffer and return the block back via virtual storage interface (MTR3). Otherwise, it reads block data from delta file in pocket storage (MTR4) and returns (MTR5) directly. Similarly, for write request, the main thread will check and update the tracking table (MTW1, main thread write #1), update the content in data block table (MTW2) and issues a pocket storage write request to pocket storage thread (MTW3). After that the pocket storage thread will handle the write requests asynchronously (PTW, pocket storage thread write). To improve the storage reliability, a similar process will take place with hard disk thread together with pocket storage thread to write block updates from memory to hard disk as backup (HTW, hard disk thread writes).

The IO throughput of USB pocket storage is slower than that of internal storages, so it impacts the overall SSPS booting and run-time performance. Based on the memory buffers design in virtual disk driver, the pre-fetch mechanism is applied to fetch the neighboring blocks as well as exact data blocks to make use of the locality of IO references. This helps reduce the frequent small data block readings from low-speed pocket storage. For writing, the asynchronous and multi-threading architecture are used in virtual disk driver during run-time to mitigate the bottleneck of IO performance gap between memory and pocket storages. When virtual disk driver is unloaded during system shut down or reboot, the in-memory data in base image buffer, data block table and tracking tables will be written back immediately, and that of backup hard disks will be erased to restore the hard disk to initial state.

E. How OS Runs on Desktop System Images

SSPS uses a special image loader and virtual disk driver to make OS run on top of virtual disk images during pre-boot period and run-time period respectively. 1) For bare-metal boot, the image loader loads system images directly. It identifies system image files from pocket storage, reads, and writes specified blocks with algorithm 1 and 2 above. When system goes to run-time period, the desktop system reads and writes block IO from system images with same algorithms by virtual disk driver. So, OS floats above system image. 2) For virtualized boot, the loader and virtual disk driver are put into hypervisor to transparently translate VMM's IO access to system image in pocket storage. Image loader and virtual disk driver separate software stack from physical storage during pre-boot and run-time period respectively.

Conceptually, SSPS host could be regarded as a light-weighted "storage virtualization" technology by moving desktop system images from native platform to a different host platform via pocket storge and recording block changes. The adaption of a single OS image to different platforms are addressed at OS level by installing multiple drivers. This is how SSPS approach solves the DGWP problem.

VI. Security Considerations

Regarding the desktop system images run across multiple hardware platforms, security issues must be carefully considered.

A. Security Assets and Potential Adversaries

The following assets during SSPS workflow should be protected. 1) original desktop system in native platforms. 2) original desktop system in host platforms. 3) the pocket storage with system images during image creating, loading, run-time updating, and merging back timeframe. As an end-to-end system, threats or damages to the assets will make the entire system not functional. Only newly added security issues brought by SSPS design are considered here. Table 2 shows the detailed information of assets and adversaries.

Compared with the software stack in native platforms, including OS and applications, SSPS approach runs software stack differently. The additional security risks explicitly lie in the creating, moving, loading, and merging back of system images across multiple platforms. However, regarding the software stack runs virtually on desktop system images, the updating of local storage is not written back to initial disk image, but copy-on-write to delta images. So theoretically, the risk of external attack to local storage is minimized to the last step of merging back. In worst case that the merging has destroyed the original native system, the initial system still has a copy in pocket storage as base image together with each recorded update delta images, so it is restorable. The basic system from pocket storage is another risk when loaded by BIOS. This would be solved by verifying in either native or host platforms. In summary, the software stack in pocket storage is loaded and run in an isolated environment - either to system or data, so the SSPS approach is safer than it appears.

TABLE 2 SECURITY ASSETS AND POTENTIAL ADVERSARIES

Security asset	Adversary	Description
Desktop system in native platforms	hostile pocket storage	The hostile software in pocket storage may attack the desktop system in native platforms.
Desktop system in host platforms	Illegal users, hostile pocket storage	The illegal users may use the system in pocket storage or other tools to attack the host platform.
The hostile software in pocket storage may attack the desktop host platform. Desktop system image in pocket storage.	Illegal users, hostile pocket storage system software	The illegal users may attack the desktop system image in pocket storage. The hostile software in native and host platform may attack the desktop system images.

B. Attack Surface

As mentioned in section V.B, there are two banks in pocket storage, the system bank with a basic system and the data bank with system images. The attack surfaces are exposed during the SSPS workflow, including: 1) user interface of pocket storage system. The basic system is a small UEFI or embedded OS environment, where the user interface such as UEFI shell or OS shell are the potential attack surfaces. 2) native and host desktop system interface, in general, the OS user interface like windows GUI. 3) network interface. Regarding the desktop system in both native and host platform is connected to network, the attack from network could be another risk. The mitigation is to disable the basic system's user interface to headless, only the three tools are allowed to run in that system and disable network interface within basic system.

C. Authentication of Pocket Storage Device

The pocket storage is loaded from native and host platforms, therefore, SSPS requires the authentication of pocket storage and system. Secure boot is an UEFI based firmware level authentication mechanism to protect the system by only loading legal loader and UEFI application using UEFI BIOS as root of trust [24], while Intel® Boot Guard provides TPM based hardware trust mechanism [25][26]. A trust-chain based authentication mechanism with secure boot and TPM is applied.

The secure boot key is installed into BIOS of both native platform and host platforms, could be optionally in hardware like TPM. When security pocket storage is plugged in and loaded, the basic system will be authenticated by BIOS as an UEFI payload. Within basic system, only the three tools could be loaded, including image creating tool, image loading tool and image merging tools. The authentication of users to run these tools is also provided. Only legal users could start the tools. As the last step of disk image creating tools, the virtual disk driver and platform adaption tools are injected to the desktop system images, an UEFI level run-time service was invoked by image creating tools to check the validity of virtual disk driver and platform adaption tools that they have not been tampered. Figure 6 illustrates the trust chain from BIOS, basic system, tools and the injected virtual disk driver and platform adaption tools.

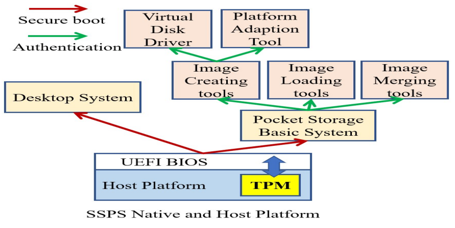


Fig. 6. Security in SSPS

D. Data Protection of Desktop System and Image

When running the desktop system image at host platforms, the host system cannot destroy or illegally snoop the guest system. Regarding the desktop system on host platform and that in pocket storage are logically separated systems, the mitigation is also to use the secure boot to authenticate the pocket storage and that of the host platforms. Additional anti-malware software can be used in desktop system image to detect the illegal attack from host storages. The certificate can be issued by a trusted certificate center.

Contrary to the protection of desktop system image in pocket storage, the desktop system image cannot destroy or illegally take data out of host platforms. In most cases, the host system is not a bare metal, but a completed native PC with both hardware and software system in local hard disks. If the desktop system image from pocket storage is loaded and run in host platforms, it's possible to access the native system storage of host platform from pocket desktop system. There are some mitigations: 1) for bare-metal mode, the local hard disk could be hidden by disabling the ACPI device table at BIOS level. 2) load the desktop system image from hypervisor directly, with which the guest desktop will completely be separated from host system. 3) check the critical OS level files to make sure they are not illegally deleted, altered, or hacked.

E. Traces Removal

After removal of pocket storage from host platforms, the end user may be worried about the risk of leaving personal data in host systems. To solve this issue, a trace-removal daemon was put into the desktop system image when it's created. When the system is shutdown at host platform, the trace-removal daemon will run as the last applications to check if there are remaining data in local storage of host platforms. In general, the temporary file, user folder, system data folder are the key places.

VII. IMPLEMENTATION OF SSPS

From the implementation perspective, creating of system image from native platform, loading and running of system image at host platform, sending images back to pocket storage and merging them to native platform are four key steps. Also, the boot and run-time performance of system image, and the performance of system image creating, moving to and back from pocket storage, and merging delta images to native platform are critical indicators to evaluate the SSPS implementation.

A. Test Bed

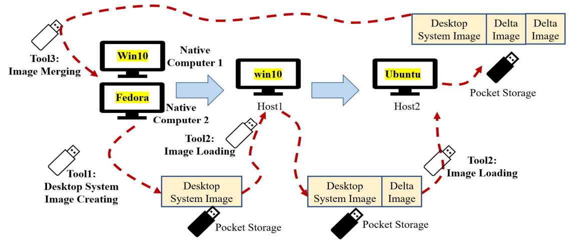


Fig. 7. SSPS Test Bed

The test bed of SSPS implementation is shown in figure 7. Computer 1&2 were native platforms with Windows 10 and Fedora respectively. SSPS host 1 was a win10 computer and host 2 was installed with Ubuntu, both as host platforms. A USB thumb was equipped with pre-boot payload and three tools: 1) image creating tool, 2) image loader tool, 3) image merging tool. The pocket storage in figure 7 was the same USB thumb to hold three tools and system image, but they are in a different partition. The detailed hardware configurations were listed below.

TABLE 3 HARDWARE CONFIGURATION FF THE TEST BED

Item	Hardware Configuration	Description
Native computer 1	Intel Core i5-8400 2.8G H310, 4G memory, Phison M.2 SSD hard disk, RTL GBE 8168E NIC, win10 x64	A win10 computer
Native computer 2	Intel Core i5-8400 2.8G H310, 4G memory, Phison M.2 SSD hard disk, RTL GBE 8168E NIC, Fedora x64	A Fedora computer
SSPS Host 1	Intel NUC10 Core i7-10710U, 6 Core 12 Thread 8GB memory, M.2 SATA 120GB HD	A win10 computer
SSPS Host 2	Intel Core i5-8400 2.8G H310, 4G memory, Phison M.2 SSD hard disk, RTL GBE 8168E NIC, win10 x64	An Ubuntu computer
Pocket storage	Thumb 1: SanDisk 128GB USB2.0 Disk CZ4 R/W 33MB/s, Thumb 2: SanDisk 128GB USB3.0 Disk CZ73 R/W 150MB/s, Thumb 3: SanDisk 128GB USB3.0 Disk CZ880 R/W 360MB/s	Three different USB thumbs for performance comparison

B. Implementation Briefing

The reason of choosing multiple native and host platforms with different OS and hardware was to test whether multiple native OSes could be taken out, transferred to different host platforms with different native OSes, and successfully work.

The USB thumb was a USB 3.0 thumb with ext4fs file system. There are two partitions in the thumb – a system partition to hold the tool 1-3, and a data partition to hold system image and delta images. The USB thumb was a self-bootable embedded Linux that was built from scratch with Linux 5.4.51 kernel, BusyBox 1.31.1 and buildroot_2020.02.1 packages. It was configured as first boot option of BIOS in native and host platform, so it would bring up USB thumb to run the three image tools in system partition.

The SSPS image loader was invoked within embedded Linux environment as the final step. It first redirected BIOS read/write system calls from native hard disk to desktop system images in pocket storage, so the OS to be loaded would assume the underlying pocket storage as local hard disk as before. When OS started to run, the virtual disk driver which was injected by image creating tool would take over all disk IO and forward to desktop system images in pocket storage. In this way, the desktop system of native platform was floated on top of images and virtually run on them.

VIII. EVALUATION OF SSPS

The boot and run-time performance, the performance of system image creating to pocket storage and sending to host platform, and merging back to native platform are critical indicators as in table 1.

A. Benchmark of Image Manipulation

Figure 8 are four performance indicators: desktop system image creating in native platform as top left, export to pocket storage as top right, import to host platforms as bottom left, and merging back to native platform as bottom right. Three different USB thumbs as table III were used for benchmark test together with different image sizes – 20G, 40G and 80G respectively. From the benchmark result, the following conclusions are drawn. 1) all four performance indicators are heavily dependent on USB throughput. For example, the fastest thumb3 may need 250 seconds to create an 80G disk image while the slowest thumb1 may need 13300 seconds, the 50X performance gap quite aligns with the write throughput of thumb1 (6MBps) and thumb3 (330MBps). As to the import of disk images from pocket storage, an 20G image may need 55 seconds and 600 seconds in thumb1 and thumb3, the 10X gap is also quite aligned with the read throughput of thumb1 (33MBps) and thumb3 (360MBps). 2) the import and export performance are nearly proportional to the image sizes. For example, during the export of 20G and 80G disk image in thumb2, the duration is about 800 seconds and 3000 seconds, both around 4X.

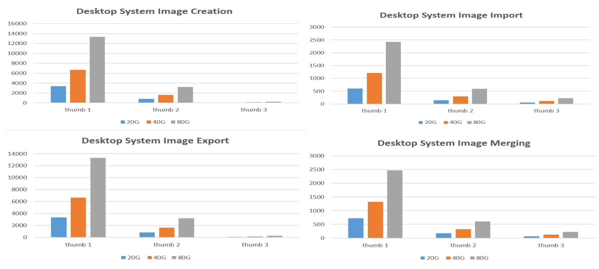


Fig. 8. Image Creation, Import & Export and Merging Benchmark

B. Boot and Run-time Performance Benchmark

As to the system booting, the left of figure 9 shows the boot of 40G win10 image in the three platforms, from nearly 31 seconds to 85 seconds compared with 24 seconds of the native booting. There is no significant performance gap between native boot and desktop system image boot from fast USB thumb. It's same as the 9G Ubuntu 20 disk image. Finally, the PassMark [27] system benchmark tools are used to compare SSPS host with native PC covering the overall, CPU, graphics, memory and storage performance. Figure 9 right is the result. For most indicators, the SSPS host's results are quite close to that of native computer, the tiny performance drop is caused by virtualization overhead.

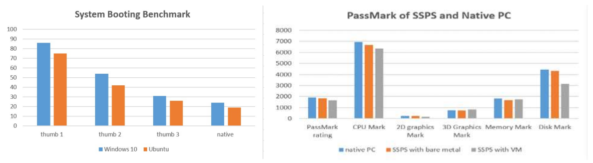


Fig. 9. System Booting and PassMark Benchmark Indicators

As can be concluded from the above benchmark data, the bottleneck of SSPS performance, either disk image creating, OS booting and merging, or the run-time performance, is USB thumb's IO performance. The hardware improvement will significantly help the overall performance increase.

IX. Challenges and Future Work

The SSPS approach solved DGWP problem by having the storage go-with-person and leveraging any available hardware to run the image. With technology moving forward, some promising technology directions come into being: 1) regarding the desktop system image becomes bigger in size and network is easier to access than before, the collaboration of local pocket storage with remote cloud would be an interesting approach. 2) the continuous performance improvement and support of more OSes will always be attractive. 3) Regarding the stricter security requirements in multiple scenarios, the exploration on performance impact brought by security policies is another direction. 4) Finally, more experiments and performance comparison with existing solutions would be carried on continuously.

X. Conclusion

The invention of PC together with its mass deployment are among the most important achievements that has changed our lives. In modern society, people are dependent on computing devices. However, it's far from convenience to bring personal computer at any time or in any places, known as the DGWP problem. Current computer software is tightly bound to the hardware. Regarding it's not difficult to find a nearby computer and bring tiny pocket storage with us, we are thinking of splitting computer software from hardware to pocket storage and run on any available hardware safely and securely. In this paper, we proposed an UEFI-based secure system-in-pocket-storage approach (SSPS) to address DGWP problem by storage go-with-person, including the SSPS workflow, host architecture, pocket storage and desktop system image, loader and virtual disk driver, system optimization and security considerations during implementation. The paper also provided system evaluation together with forecasting of promising technologies along this direction.

XI. Acknowledements

We thank all anonymous reviewers for the detailed reviews and valuable feedback. The technologies in this paper were supported successively by NEX NSWE, IOTG HEC and IAGS UEFI organizations of Intel Corporation.

References

[1] Wikipedia, History of Laptops, [Online]. Available: https://en.wikipedia.org/wiki/History_of_laptops

[2] Intel, Intel® Compute Stick STK2M3W64CC STK2MV64CC STK2M364CC Technical Product Specification, [Online]. Available: https://www.intel.com/content/dam/support/us/en/documents/boardsandkits/computestick/STK2m3W64CC_STK2mv64CC_STK2m364CC_TechProdSpec.pdf

[3] Dunhui Yu, Jian Wang, Bo Hu, Jianxiao Liu, Xiuwei Zhang, Keqing He, and Liang-Jie Zhang, A Practical Architecture of Cloudification of Legacy Applications, 2011 IEEE World Congress on Services

[4] Vmware, Virtual Desktop Infrastructure (VDI), [Online]. Available: https://www.vmware.com/topics/glossary/content/virtual-desktop-infrastructure-vdi

[5] Citrix, VDI and DaaS, [Online]. Available: https://www.citrix.com/solutions/vdi-and-daas/what-is-vdi-virtual-desktop-infrastructure.html

[6] Pete Batard, Rufus, [Online]. Available: https://rufus.ie/en/

[7] S. Potter and J. Nieh, "Highly Reliable Mobile Desktop Computing in Your Pocket," 30th Annual International Computer Software and Applications Conference (COMPSAC'06), 2006, pp. 247-254, doi: 10.1109/COMPSAC.2006.52.

[8] B Gilbert, A Goode, M Satyanarayanan, Pocket ISR: Virtual machines anywhere, reports-archive.adm.cs.cmu.edu

[9] Steven Osman, Dinesh Subhraveti, Gong Su, and Jason Nieh, The Design and Implementation of Zap: A System for Migrating Computing Environments, Appears in Proceedings of the 5th Operating Systems Design and Implementation (OSDI 2002), Boston, MA, December 2002

[10] Jacob Gorm Hansen and Eric Jul, Self-migration of Operating Systems, Proceedings of the 11th workshop on ACM SIGOPS European workshop, September 2004 Pages 23–eshttps://doi.org/10.1145/1133572.1133616

[11] Haikun Liu, Hai Jin, Xiaofei Liao, Chen Yu and Cheng-Zhong Xu, Live Virtual Machine Migration via Asynchronous Replication and State Synchronization, IEEE Transactions on Parallel and Distributed Systems (Volume: 22, Issue: 12, Dec. 2011)

[12] Shripad Nadgowda, Sahil Suneja, Nilton Bila, Canturk Isci, Voyager: Complete Container State Migration, 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)

[13] Bo Xu, Song Wu, Jiang Xiao, Hai Jin, Yingxi Zhang, Guoqiang Shi, Tingyu Lin, Jia Rao, Sledge: Towards Efficient Live Migration of Docker Containers, 2020 IEEE 13th International Conference on Cloud Computing (CLOUD)

[14] R. Bez, E. Camerlenghi, A. Modelli and A. Visconti, "Introduction to flash memory," in Proceedings of the IEEE, vol. 91, no. 4, pp. 489-502, April 2003, doi: 10.1109/JPROC.2003.811702.

[15] WikiPedia, Flash Memory, [Online]. Available: https://en.wikipedia.org/wiki/Flash_memory

[16] Yaoxue Zhang, Kehua Guo, Ju Ren, Yuezhi Zhou, Jianxin Wang, Jianer Chen, Transparent Computing: A Promising Network Computing Paradigm, IEEE Computing in Science & Engineering (Volume: 19, Issue: 1, Jan.-Feb. 2017)

[17] Yaoxue Zhang and Yuezhi Zhou, Transparent Computing: Spatio-Temporal Extension on von Neumann Architecture for Cloud Services, TSINGHUA SCIENCE AND TECHNOLOGY ISSNll1007-0214ll02/12llpp10-21 Volume 18, Number 1, February 2013

[18] UEFI.org, UEFI Specification, Version 2.9, [Online]. Available: https://uefi.org/specifications

[19] M. Wu, "Analysis and a Case Study of Transparent Computing Implementation with UEFI," Int'l J. Cloud Computing, vol. Volume 1, no. Issue 4, 2012, pp. 312–328.

[20] Intel whitepaper, Aug 2020, Document Number: 630941, "Intel® Transparent Client Infrastructure Technology"

[21] M. Wu, L. Zhou and F. Huang, "EVCS: An Edge-assisted Virtual Computing and Storage Approach for Heterogeneous Desktop Deployment," 2022 IEEE 8th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), 2022, pp. 107-112, doi: 10.1109/BigDataSecurityHPSCIDS54978.2022.00029.

[22] Flexense, USB3 vs. SATA Disk Performance Comparison, [Online]. Available: https://www.flexense.com/usb3_vs_sata_disk_performance_comparison.html

[23] Cnet, Reliability of USB flash drives questioned, [Online]. Available: https://www.cnet.com/tech/computing/reliability-of-usb-flash-drives-questioned/

[24] NIST, Root of Trust, [Online]. Available: https://csrc.nist.gov/Projects/Hardware-Roots-of-Trust

[25] Trusted Computing Group, TPM, [Online]. Available: https://trustedcomputinggroup.org/tpm-2-0-library-specification-approved-isoiec-international-standard/

[26] Intel, boot guard technology, [Online]. Available: https://edc.intel.com/content/www/us/en/design/ipla/software-development-platforms/client/platforms/alder-lake-desktop/12th-generation-intel-core-processors-datasheet-volume-1-of-2/001/boot-guard-technology/

[27] PassMark Software, Easy PC Benchmark Tools, [Online]. Available: https://www.passmark.com/products/performancetest/