Development of a SCION router towards a secure Internet

Bernardo Conde
Email: bernardoconde@tecnico.ulisboa.pt
Instituto Superior Técnico

I. Abstract

Today's Internet architecture, while a marvel of software engineering, is plagued with serious security issues, mostly due to some of its early design decisions. These issues are extremely hard to fix with incremental solutions, prompting the question of whether we should design a new, future Internet architecture, capable of delivering the same service we have grown to love, without all the security problems it entails. SCION is a new Internet architecture with a focus on security and availability. SCION's clean-slate approach means that its architecture is not compatible with the current network infrastructure. To gain traction, it is therefore fundamental to address a hard challenge: SCION has to achieve at least the same level of packet processing performance as current Internet routers. As a step in this direction, in this work we present an implementation of a terabit speed SCION data plane router. We implement our solution in a state of the art network device, a Programmable Switch, powered by an Intel Tofino ASIC. These devices allow for the definition of their network functions to be done in software, using the P4 programming language. However, in order to achieve line-rate speeds, their computation model is restrictive, exacerbating the challenge. We show that our solution is able to achieve terabit per second line-rate speeds in our target, while performing the required on-demand cryptographic operations, which represents more than two orders of magnitude speedup over the state of the art.

II. Introduction

Today's Internet is an undeniable success, having completely revolutionized global communications and created multiple indispensable services, becoming the backbone of our modern, everyday life. However, the current Internet architecture was conceived 50 years ago, without any expectations that it would achieve the success that it has. As such, some of its early design decisions have caused multiple severe limitations, particularly with respect to security.

To make matters worse, today's Internet architecture suffers from an "ossification" of its network layer, due to the forceful use of the Internet Protocol [19]. IP allows hosts to forward packets to one another. Unfortunately, due to its design, it makes the paths each packet takes completely opaque to its hosts, not allowing any control over the path to use nor to know which path a packet took towards its destination. Another issue is that it forces routers to maintain state, namely forwarding tables, making it more expensive to maintain, evolve, and scale the network infrastructure. Finally, IP packets are not authenticated, allowing a malicious host to lie about its identification, spoofing an IP address of another host. This enables possible Denial-of-Service attacks that have caused serious disruptions [26].

Another "ossified" protocol is the inter-domain control plane protocol of the Internet, the Border Gateway Protocol [27] (BGP) . This protocol is responsible for deciding how packets get routed across the Internet, through the exchange of routing and reachability information. Unfortunately, due to the lack of authentication in this exchange, the protocol is susceptible to many security attacks that, when exploited, can result in interception of users' information or the complete blackout of essential services. While there have been extensions to this protocol [20, 17] that address some of these issues, these solutions also come with problems of their own.

In order to fix these issues, a new secure-by-design Internet architecture, called SCION [18], has been proposed and created. SCION is a "path-aware Internet architecture, designed to provide complete route control, failure isolation, and explicit trust information for end-to-end communications". An advantage of SCION is the clean separation between the data plane protocol (the part of a network that carries user traffic) and the control plane protocol (the part of a network that carries signaling traffic and is responsible for routing) that it uses. Because of this separation, it is possible to have different protocols in each plane, while still respecting the SCION architecture. The SCION data plane is currently EPIC [13], a family of protocols focused on increasingly and incrementally adding stronger security guarantees. It has 4 versions, from EPIC L0 (the minimum offer of security) to EPIC L3 (the most secure, offering all the security guarantees from EPIC L0 to L2, plus others). These guarantees will be detailed in Section IV-A.

III. Motivation

SCION is a production-ready Internet architecture, currently offered by 12 SCION-native Internet Service Providers [8]. However, for SCION to be even more broadly adopted, it is necessary for its data plane to deliver similar performance to today's Internet. Recent work [25] has implemented an initial prototype of EPIC L0 in hardware, to an FPGA target, achieving 40 Gbps throughput. In this work we move forward in three fronts. First, we will design and implement a SCION-compatible router data plane in a programmable switch, with

the goal of increasing the data plane performance by more than two orders of magnitude. Our specific target is the Intel Tofino ASIC, capable of 12.8 Tbps. Second, we will implement both the EPIC L0 and the (more secure) L1 protocol versions. These protocols offer more security guarantees than SCION, fixing possible attack vectors. Our implementation will be done fully on the data-plane of the programmable network devices, allowing it to run at terabit speeds. Finally, we will implement our prototype using P4 [4], a domain-specific language for packet processing that can be compiled to various targets, from SmartNICs to programmable switching ASICs. Programming in P4 will facilitate porting to other targets.

In this work we will:

- Design and implement a SCION-compatible router, running the EPIC L1 protocol fully on the data-plane, which offers better security guarantees than regular SCION. We will target a programmable network switch, capable of terabit speeds and we will implement our router using the P4 programming language.
- In order to achieve this, our router will need to perform one cryptographic operation per-packet. Due to limitations of previous work, we will propose and adopt a lightweight cryptographic construct, the Simplified Even-Mansour. We will implement this construct for our target in an efficient manner, so our implementation is able to achieve the required throughput
- Finally we will evaluate our implementation against the state of the art and show that, in all cases, we are able to achieve comparable or better performance, in terms of throughput, while providing better security guarantees

IV. Background

The Internet is a global, decentralized network, comprised of tens of thousands of interconnected networks, or AS. Each AS is normally under the control of a single administrative entity. Information travels between ASes using one out of many paths, chosen by a routing protocol, which exchanges information about the reachability of every destination.

While the rapid development of Internet services and technologies make it clear that the Internet is progressing quickly, the same is not true of its architecture. While its top (Application and Transport) layers and bottom (Datalink and Physical) layers have evolved significantly, the Internet middle stack has stagnated for decades (shown in Figure 1): both the data-plane and the inter-domain routing have been dominated by two protocols, IP [19] and BGP [27], respectively.

Flaws with IP The Internet Protocol, or IP, is one of the fundamental protocols in the current Internet architecture, responsible for the forwarding of packets between hosts. For this, each sender only needs to know the receiver's address, which will then be written on each of the packets' header and disseminated through the network.

Unfortunately, while this approach is simple, it has many drawbacks. One of these drawbacks is the fact that the path for a packet is opaque both for the sender and the receiver. Due to a lack of specification of any path in a packet, neither the sender nor the receiver have any ability to influence the path a specific packet can take, nor the information to know which path a packet took. This makes it impossible to prevent a packet from being routed through non-trusted networks, or to choose a more suitable path through a desired metric.

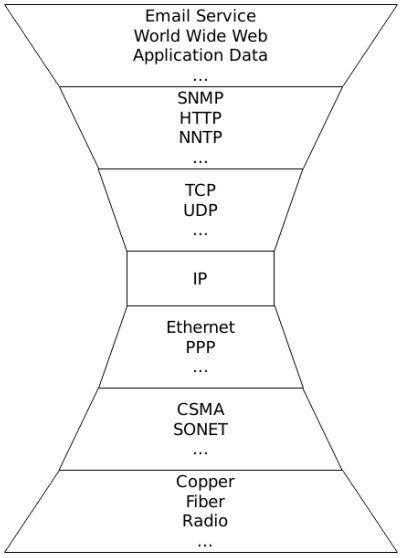

Fig. 1. The Internet Hourglass Problem: while application-level protocols and physical level protocols can be changed/upgraded very easily, today's Internet architecture has a very hard dependency on its network-level protocol, the Internet Protocol, making it extremely hard to fix its problems or replace it with another, better protocol.

Another serious issue is the fact that routing and forwarding are bound together, since packet forwarding depends on the state of forwarding tables in routers, which can change over time. This means that a working path can change or even break after an update to the forwarding tables' state, which can occur at any point, potentially causing serious connectivity issues.

Additionally, due to each router's necessity to maintain forwarding tables, and to perform a lookup for each packet, each router's hardware must have special memory constructs to deal with these constraints. These components are very expensive and energy-intensive, leading to an increase of hardware's expenses. Plus, various Denial-of-Service attacks can come from attackers that exhaust a router's memory by filling forwarding tables maliciously [3].

Finally, IP lacks any security guarantees over routing and packet deliverance: a packet can pass through any intermediate hop at any time due to its lack of path authentication. This makes enforcing the use of specific paths for certain hosts extremely challenging.

Flaws with BGP Unfortunately, while BGP works well most of the time, one of its main limitations is security [5]. In BGP, the control plane and the data plane are not cleanly separated: each AS advertises its reachability information about its list of IP prefixes as a BGP UPDATE message that is sent using the data plane. This lack of separation often causes packet forwarding to fail when a valid route changes (due to the acceptance of a BGP UPDATE), seriously affecting the underlying traffic [12].

Another issue with BGP is the lack of fault isolation: since

it is a completely distributed system with no hierarchy or isolation, any BGP client can affect the whole network, which means that any client can seriously disrupt global connectivity. As an example of this, in 2008 a single wrongly-configured router from Pakistan Telecom made YouTube unreachable to the whole world for hours [21]. Because each update needs to be sent to every AS as well, BGP tends not to scale very well, and due to the time it takes to disseminate a message through the whole network, a consistent global view of all correct and available routes can take several minutes, which can provoke service outages for users (in fact, it has been shown that for certain situations, BGP may never fully converge [11] or converge non-deterministically [10]).

BGP also only selects a single path, providing no multi-path support. This can lead to bottlenecks when BGP selects a legitimate but inefficient route through a congested link. Because of the inability of the end hosts to choose their own path for their packets to take, they have no choice but to wait until ASes in the Internet modify policies such that a more appropriate path gets chosen.

Moreover, BGP performs no authentication nor authorization of paths, allowing a malicious Autonomous System to perform a hijack (as the Pakistan incident above) or interception attack, giving a malicious entity the ability to passively monitor and collect traffic, or to create a blackhole.

To address these and other security problems, two mechanisms, the RPKI [20], and BGPsec [17] have been proposed. These standards allow an AS to cryptographically sign a BGP route announcement, thereby making the above attacks impracticable.

However, the RPKI and BGPsec do not solve all problems. For instance, while the RPKI offers origin authentication and thus works against the IP hijack attacks, it still allows other, more sophisticated attacks to occur [15]. For example, a malicious AS trying to hijack a particular IP prefix can still send a BGP UPDATE message claiming that it is *directly connected* to its legitimate owner. Recipients of such an announcement would accept it as the legitimate owner of the addresses, as the malicious AS is noted as the last AS in the BGP UPDATE and would then start sending the traffic destined for those IP addresses to the attacker, who can then inspect, reroute, or drop it.

BGPsec addresses this issue by signing the entire path, but attackers are still able to create so called "wormhole" attacks [14] (two ASes conspire with each other to generate valid BGPsec signatures, in order to convince victim ASes to use them for communication) and cause forwarding loops.

BGPsec has also been reported to work poorly unless all ASes use and enforce BGPsec [16]. When used in a partial deployment scheme (only some ASes enforce BGPsec, while others do not), BGPsec can lead to severe issues like instability. Due to backwards compatibility requirements, BGPsec is also prone to downgrade attacks (where attackers fake not supporting BGPsec in order to more easily attack the network).

Another problem with BGPsec is the existence of circular dependencies [7]. In order to be able to participate in the network securely, one must be able to fetch and exchange cryptographic keys and RPKI certificates. In turn, in order to fetch these, one needs to already be able to participate in the network. This creates a circular dependency.

Another problem is the ability of enforcement of network sovereignty, as organizations at the root of the RPKI hierarchy have the power to create or revoke certificates. Depending on the jurisdiction, local courts of some of the countries of origin for these organisations may gain the power to shut down parts of the Internet (with the obvious possibility of abuse).

Finally, BGPsec exacerbates BGP scalability issues. Under BGP, in order to provide global connectivity, every single global AS in the world needs to know how to reach every other AS. This requires a large number of BGP UPDATE messages, the processing of which requires much more resources in BGPsec, due to the additional cryptographic checks. Furthermore, prefix aggregation, which is used to combine multiple IP prefixes to reduce the number of routes and announcements, no longer works in BGPsec. This is particularly cumbersome as the increasing fragmentation of the IP address space and the trend towards announcing ever smaller IP address ranges have caused a strong growth of the number of paths that internet routers need to store and exchange. All these problems have hindered the widespread adoption of BGPsec.

A. SCION: Secure Future Internet architecture

SCION [18], short for Scalability, Control, and Isolation On Next-Generation Networks, is a new, clean-slate, path-aware, Internet architecture, aiming at "offering complete route control, failure isolation, and explicit trust information for end-to-end communications". SCION groups existing Autonomous Systems into ISD that connect with each other to provide global connectivity (Figure 2). Each ISD is administered by a subset of its ASes, called the ISD core, which is responsible for setting the ISD policy. These ISD cores establish the available paths for communication inside and outside the ISD (this is called the path-exploration process), which will be disseminated through all of the hosts of each ISD. These hosts will then specify which path to use for each packet, by making them explicit in the packet's header. Every packet sent outbound the ISD must pass through the ISD core, allowing each ISD to set strong forwarding policies and to prevent malicious path creation. It also enables defenses against network attacks: since the full path of a packet is carried in the packet itself, finding out the original sender of a packet is possible, allowing the receiver and the network to deal with them accordingly. SCION also completely separates the control plane from the data plane, ensuring that forwarding cannot retroactively be influenced by control plane operations. For the control plane, SCION uses SCMP, analogous to ICMP. It provides functionality for network diagnostics, such as $ping$ and $traceroute$, and error messages that signal packet processing or network-layer problems.

EPIC EPIC [13] is a family of data plane protocols designed to be used in path-aware internet architectures like SCION. It presents 4 different versions, each designed to provide increasingly strong security properties, while allowing: network operators to be able to impose their own policies; end hosts to verify that their forwarding decisions are followed

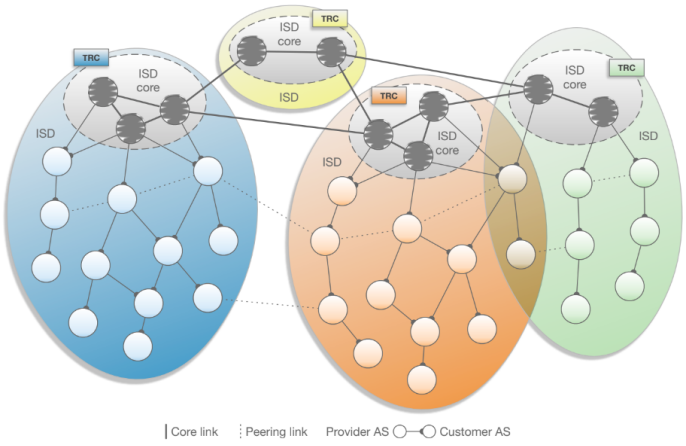

Fig. 2. SCION ISD model (from [18]): Multiple different ASes that trust each other join together, becoming ISDs. For each ISD, there is a group of special ASes, called the ISD core, responsible for enforcing rules over all the traffic sent to other ISDs. These ISD cores also are responsible for creating and disseminating the authorized paths to each of their own ASes, as the ISD policy seems fit.

by the network; and intermediate routers and recipients to authenticate the source of packets.

The EPIC family of protocols is made up of four different versions:

- EPIC L0: EPIC L0 is a simplified version of the original data plane protocol used in SCION. During the path-exploration process, each AS generates a Hop Authenticator: a static MAC, using each AS key, over the paths's starting timestamp, the hop information, and the previous Hop Authenticator, truncated to len_{ha} bytes. When each source obtains a path, including all Hop Authenticators for that path, it will send their Hop Validation Field as just each Hop Authenticator; every intermediate router checks if the packet came from the correct interface and if the Hop Validation Field corresponds to the correct Hop Authenticator. L0 has a clear problem: if the Hop Authenticator length len_{ha} is not big enough, its security properties can be broken with an online brute-force attack, generating valid Hop Authenticators for invalid paths. Since Hop Authenticators will be used as Hop Validation Fields, and these Hop Authenticators can be reused for different packets, one only needs to do this costly attack once.
- EPIC L1: EPIC L1 solves the brute-force attack problem by making each Hop Validation Field dependent on packet timing information, thereby making sure that they cannot be reused. Each Hop Validation Field is a MAC of the packet timestamp, plus source and the host address, using the Hop Authenticator of each AS as a key.
- EPIC L2: EPIC L2 extends the previous security guarantees by also allowing intermediate routers to authenticate the source of a packet and the destination to authenticate its payload. Each intermediate AS generates a new key, derived from the source host and AS information. The source host will then generate, for each Hop Field Value, a MAC using the same information as L1, but using this new, generated, AS-unique key. For the destination, another new key will also be generated, based on the source and destination's host and AS, which will then be used by the source to create a MAC of the contents of the packet, plus information about the path. Since all the information to derive these keys is public, all keys can be locally generated.
- EPIC L3: Finally, EPIC L3 provides the strongest security properties, adding also the ability for both the source and the destination to perform path validation. For that purpose, each on-path AS overwrites their Hop Validation Field with a proof that they have processed the packet. This proof is the higher part of the non-truncated MAC used to generate each Hop Validation Field. Since, at the start, these fields contain the lower part of the MAC, truncating it so it fits, and since this is information that both the source and the destination already have, both can check that each packet has passed through each AS. The destination gets this updated information right when it receives the packet; in order for the source to get it, the destination sends this information as an EPIC L2 packet, in order to prevent cyclical confirmations.

TABLE I
SECURITY GUARANTEES PER EPIC PROTOCOL VERSION

Version	Path Authorization	Freshness	Packet/Source Authentication	Path Validation
L0	Yes	No	No	No
L1	Yes	Yes	No	No
L2	Yes	Yes	Yes	No
L3	Yes	Yes	Yes	Yes

Overall, EPIC is a family of protocols that offers strong security guarantees, while being lightweight enough that we expect it may be possible to run it fully on a fast data plane, such as a modern SmartNIC or a programmable switch.

B. Programmable switching ASICs and the P4 programming language

Traditional routers and switches are *fixed-function*: the way they process packets is fixed during the chip design phase. A new class of programmable switching ASICs has recently emerged that allows packet processing to be specified by a program, and to be reconfigured in the field, improving the flexibility of modern networks [1]. The architecture of switches tends to follow a multi-stage pipeline process, where different stages of the pipeline look and act on different header fields of each packet. While this process tends to add end-to-end latency to each packet, it also allows for multiple packets to be processed simultaneously, since different stages of the pipeline can run concurrently in different packets.

The pipeline of a programmable switching ASIC is often refereed to as the Protocol Independent Switch Architecture, or PISA (see Figure 3). This architecture starts by running a programmable parser on each packet, responsible for recognizing the header fields and matching them to later stages. Then, a sequence of Match/Action rules runs over all matched

packets, tasked with potentially acting upon one or more of the identified header fields. Finally, there is a deparser, that serializes the packet metadata, obtained from all the in-memory header fields processed by earlier stages, into the packet, in order for it to be transmitted into the output link.

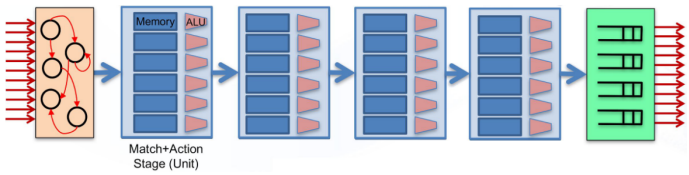

Fig. 3. PISA is made of 3 steps: the parser (red), a pipeline comprised of multiple Match/Action units (blue) and finally a deparser (green). In order to achieve fast speeds, the computations in each Match/Action unit are very constrained.

P4 In recent years, the Programming Protocol-Independent Packet Processors [4] language, better known as P4, has received a lot of support, becoming the *de-facto* way of programming programmable data planes, including ASICs, FPGAs, SmartNICs, and even x86 software switches. P4 is a domain-specific language, custom-made for programming the data-plane functions of programmable network devices.

A P4 program is structured into several different components:

- Headers: The specification of the header formats, as structures with named fields and their lengths.
- Parsers: A finite state machine defining how to parse each header into one of the previously defined structures.
- Tables: Description of multiple match/action rules, and which actions are to be performed for each match.
- Actions: Set of primitive functions to build more complex sequences of actions.
- Control Programs: Specification of the order that match/action rules shall be applied for a packet, defining the control flow of the packet processing.

Each packet processing unit starts with the processing of arriving packets by the parser, which extracts each header. These header fields are then sent through a series of match/action tables divided in two parts: the first is the ingress pipeline, that determines the egress port and queue order of the packets, besides other functions; and the egress pipeline, responsible for defining the destination ports and number of instances of the packet to send, besides other programmer-defined actions, again.

The flexibility of P4 targets lends themselves as viable targets to become the infrastructure elements of a new Internet architecture, as it allows the precise definition of the packet processing of the network data plane.

V. Previous Work

This section discusses previous implementations of SCION compatible data planes. To the best of our knowledge, there are currently three implementations of the SCION data plane: the official, software-based, x86_64 implementation [24]; a NetFPGA-based one by Soucková [25]; and a P4-based one, targeting a Intel Tofino, by Joeri and Schutijser [22]. In this section, we will present them one by one, and analyze their advantages and disadvantages.

A. x86_64 SCION official implementation

The SCIONLab institution offers a free, open-source, complete implementation of all the SCION protocols [24], including a full specification of designs. Our solution follows this specification, in order to be compatible with existing software. This implementation is written in the Go programming language. It is only a reference implementation, useful for testing purposes. It is not intended for use in large-scale networks, as its performance is limited.

B. NetFPGA SUME hardware implementation

The NetFPGA SUME [28] is an FPGA-based PCI Express board with I/O capabilities for 100Gbps operation, capable of running P4-based programs. In 2019, Soucková [25] presented an implementation of the SCION data-plane, very similar to EPIC L0, running on a NetFPGA SUME. By reconfiguring the internal FPGA into an extremely optimized AES module, they showed parts of an implementation processing packets at 40Gbps, in a device with 4 10Gbps ports.

The most important part of this design was the fact that the cryptographic operation was ran *online*: for every packet, a MAC was computed by the FPGA module and checked with the one contained in the packet, for authentication purposes. This work was extremely important in showing that an implementation of EPIC L0 that performs its cryptographic operations per packet is possible.

The main drawback of this design is its throughput: in this day and age, 40Gbps is not enough for a production-ready router. Also, this design implements the original SCION data-plane protocol, that is vulnerable to MAC brute-force attacks. Our objective is to improve the performance of these metrics by at least two orders of magnitude, while making it more secure against this type of attacks.

Worth of note is that the implementation failed to be completed due to latency issues: in order to be able to process packets at line-rate, they were required to have the entire processing pipeline fit in a 5ns window; however, their only able to fit their fastest implementation in 5.08ns. However, parts of it were still able to be evaluated independently. We will use these independent results in the evaluation against our solution.

C. Intel Tofino implementation of EPIC L0

Recently, Joeri and Schutijser [22] presented an implementation of the SCION data-plane protocol running on an Intel Tofino. In order to be able to run on the Tofino, they were forced to do two things: the first was to pre-compute all cryptographic operations; the second one was to heavily refactor the headers of a SCION packet in order to be easier to parse.

Since AES is too expensive to be done on the data-plane of a Tofino, Joeri and Schutijser had to move all the cryptographic operations to outside the data-plane. Instead,

during path discovery, the general CPU on the device would be responsible for performing all the cryptographic computations, precomputing a MAC for every possible path and storing them in a table, accessible by the data-plane. Since, on EPIC L0, each MAC only authenticates information related to a path, this allows the cryptographic operation for each packet to become a simple table read and comparison operation. If, for a specific path, the MAC on the precomputed table is them same as the one in the packet, then the packet is authenticated and can pass through.

The main advantage of this implementation is its data plane throughput: due to running in a programmable switch, this implementation is capable of running at 12.8Tbps, with each port running at 100Gbps. Another advantage is its portability: as long as a switch implements the same open-source interface as the Tofino, porting to it is a simple matter of compiling the P4 program.

However, there is a drawback: due to the fact that the cryptography is all precomputed *offline*, it is *impossible* to implement the more secure EPIC L1 protocol. For EPIC L1, the MAC not only protects the path but also the timestamp in the packet. While this protects against brute-force attacks, it also means that it is impossible to precompute all possible MAC values.

VI. Proposed Solution

The following section describes the implementation of a SCION data plane router, running on Tofino-enabled programmable switches, implemented in the P4 programming language. First, we expose the challenges of implementing cryptographic primitives targeting a programmable network switch. We then present our solution to these challenges, and the resources used by our implementation of these primitives. Finally, we give an in-depth overview of our data plane design and implementation, describing each of its pipeline's functions.

A. Fast Cryptography in the data-plane

Unfortunately, in order to be able to process packets at 12.8 Tbps, the computational model of P4 programmable switches is extremely limited. In these pipelines, we are only allowed to do a very small number of single operations (very small number of ALU operations and small number of table accesses, typically limited to one read/write operation per stage). It is therefore very challenging to implement on-demand cryptography, namely running one full MAC operation per packet in one pipeline pass. In order to execute an expensive cryptographic scheme, like AES, in the data-plane of one of these devices, we would be forced to recirculate packets: pass the packet through the pipeline multiple times until the computation is complete. While this would eventually get us a valid MAC, it suffers from a performance penalty as recirculation limits throughout. For example, the fastest current implementation of AES in the data-plane on these devices can only output less than 11 Gbps [6].

Our solution was to implement a different, but more lightweight cryptographic scheme, that can run in one full pass through the pipeline: the Simplified Even-Mansour scheme.

B. The Simplified Even-Mansour scheme

The Simplified Even-Mansour scheme [9], or SEM, created by Orr Dunkelman, Nathan Keller, and Adi Shamir, is a one round Even-Mansour scheme where both the pre-whitening key (key applied before the permutation) and post-whitening key (key applied after the permutation) are the same:

$$SEM(M) = (P1(M \oplus K) \oplus K) \tag{1}$$

where $P1$ is a N-bit permutation.

The maximum security upper bound it can offer is $\frac{2^n}{D}$, where D is the number of known ciphertexts. With $n = 128$, we can offer up to 2^{80} security if the attacker has at most D = $2^{40} = 1.0995116e+12$ known ciphertexts. We chose this as it is the simplest possible construction of a block cipher which has a formal proof of security.

As for the n-bit permutation, we decided to implement a Substitution-Permutation-Substitution layer where each substitution is different depending on the byte position. In order to implement the scheme efficiently in a programmable switch, we made two important design decisions.

The first was that all operations are done at the byte level. While the P4 Language Standard demands support for bit-slice operations, due to the design of the Tofino's ALU, operations on data with size different than a multiple of 1 byte are extremely costly. Not only this, but operations on bit-slices can introduce data dependencies. These data dependencies can block possible optimizations and instruction reordering from the Tofino compiler, which would lead to an increase on the number of stages necessary due to the need for more computations, resulting in an inefficient implementation. Due to this, we decided to implement all the scheme's operations entirely at the byte level, parsing bigger words into smaller, byte-sized variables. Each SBOX and the permutation layer also act only at the byte-level, allowing the Tofino compiler to further optimize the code at each stage.

The second was that all SBOXes were implemented as statically allocated P4 tables. This allows us to translate this substitution primitive as a main function of the Match/Action units, making it extremely efficient. In order to guarantee even more data-parallelism, we also instantiate multiple copies of each SBOX, so that every table lookup is independent of all other lookups.

With these two methods, we are able to fit this scheme into the full pipe-line of the Tofino programmable switch, and only use 8 stages (of the 12 available in our switch).

C. SCION EPIC L1 entirely in the data-plane

The proposed solution is implemented on top of a programmable switch, with a Tofino ASIC, using the P4 programming language. In order to be fast, the router only deals with well-constructed, valid, packets. Since the main purpose of this work is to show the viability of implementing EPIC L1 in a programmable switch, features like error generation are not implemented: if a packet is invalid, for whatever reason (bad parsing, wrong MAC, invalid path, etc.), the packet is sent directly to the general-purpose CPU where it can then be dealt with accordingly.

The router is divided into 3 parts: the packet parser, which parses the packet into P4 data structures, while checking for correctness in its fields; the MAC generator, that takes the data from the packet, generates a MAC and compares it with the MAC from the packet to authenticate it; and finally the packet header fixer, that updates the packet header values so the next-hop can process the packet correctly, according to the EPIC protocol. These last two are fully implemented in the Ingress part of the Tofino ASIC.

An overview of the protocol implemented is as follows:

1) Parse the incoming packet according to the EPIC header structure. In case of error, send to the CPU.
2) Extract and parse both the current hop field and the previous hop field headers. If we are the originating hop field, we only parse this one. Otherwise, parsing the previous hop field is necessary since its MAC will be used in the data block for our own MAC verification.
3) Verify that the ingress interface in the packet corresponds to the port it originated from. The path exploration phase records which ASes are connected to each of our ports. If we receive a packet from an AS connected to a different port than recorded, send to the CPU.
4) With all of the information parsed, compute the MAC over the packet data. Check if the computed MAC matches the MAC present our packet hop field. If it does not match (meaning that the MAC in the packet is incorrect), send to the CPU.
5) If the destination AS is our AS, then forward it to the correct port, accordingly to intra-AS rules stored in the switch's tables.
6) If the destination AS is not our AS, we are an intermediate hop. We update the packet's values, so the next AS can correctly parse the fields related to it.
7) Forward the updated packet to the next AS, identified through the ingress interface in the packet.

Packet Parser The packet parser is implemented as a state machine, where each state parses specific header fields. The trickiest part of implementing the parser was parsing the $Info$ and $HopField$ headers (part of the SCION header specification [23]), since their number is variable and depends on previous header information. While P4 has support for variable size fields, high speed P4 targets as the Tofino switch do not support them given its complexity, and the requirement of guaranteeing line rate performance. For parsing the $Info$ headers, we used multiple different states in the parser state machine, one for each possible number of fields (3 states in total). However, for the $HopField$ headers, this is not a suitable solution. The maximum number of $HopFields$ in a header is limited to 64. Adding 64 different stages only for parsing these structures would make the parser inefficient and clutter the code. Due to this, we decided to exploit a design decision of SCION. In SCION packets, we only need to access, at most, two $HopField$ headers: the current header, and, if it exists, the last header before the current one. This is because these two headers contain all of the information necessary to perform the MAC computation. While we must include the full hop field information in every packet we process, we don't need to parse them all into individual structures, since we won't be accessing their data. Because of this, we can parse all of the unimportant $HopField$ headers into large-size buffers, and jump over them. We only need to be careful to correctly parse the two above-described headers into their correct structures.

MAC Generation and Authentication The MAC authentication is ran during packet processing and consists of generating a full MAC, with the Hop Authenticator as a key, based on a 128-bit block, constructed with data present in the packet. Each router computes the MAC for its own hop, and compares with the MAC value present in its Hop Field header. As for the MAC construction, we do a full round of the Simplified Even-Mansour scheme, described in VI-B, on a 128-bit block.

The main difference between MAC construction between EPIC L0/previous SCION implementations and EPIC L1 is the presence of a new packet timestamp, generated when this packet is sent. This timestamp, when included in the data block used by the MAC function, acts as a freshness value, assuring that MACs are not only dependent on the path a packet will take but are also dependent on this timestamp. This means that an attacker can not brute force a valid MAC for one specific path, since if the timestamp in the packet is too old, a hop can simply choose to discard it. This forces the attacker to update the timestamp in the packet, which in turn changes the MAC value.

The EPIC paper [13] suggests small MAC sizes, truncating the full computation to 3 bytes per hop, in order to diminish the necessary overhead in packet header sizes. However, in order to have stronger guarantees against brute-force attacks, the original SCION header specification [23] mandates MAC sizes of 6 bytes. In order to be more compatible with the SCION protocol, we only support 6-byte MAC fields.

Fixing Packet Header Values Finally, if the packet was processed correctly, we now must send it to the correct port. However, if we are an intermediate hop, we must also update the packet header values to reflect that we have successfully validated the packet and to make it possible for the next hop to find its own correct header fields.

VII. Evaluation

In this section, we present the evaluation of our solution. We compare our own implementation of the EPIC L0 protocol against our implementation of the EPIC L1 protocol, which was the main contribution of this work, and show that there are no meaningful differences in performance between the two, and that both are able to process packets at line-rate. In addition we compare our solution to the previous state of the art, and show that we are able to maintain or surpass the throughput of previous solutions, while providing the stronger security guarantees that EPIC L1 offers.

A. Comparison between EPIC L0 and EPIC L1

We compare our implementation of EPIC L0 with our implementation of EPIC L1, both targeting a programmable

switch backed by an Intel Tofino. We evaluate them both on resource usage of the Tofino chip (how many stages each use, how many resources they use on each stage through the PISA pipeline), and on throughput. We conclude by showing that resource usage of both implementations is similar, with both are able to achieve terabit speeds on existing hardware.

Resource usage We start by asking one question: do we need more resources to implement EPIC L1 versus EPIC L0. Due to the additional security guarantees that EPIC L1 offers (freshness), our expectation was that EPIC L1 would consume more resources than EPIC L0. However, this is not the case: our implementation of EPIC L1 has very similar resource usage than EPIC L0 (see Table II). They use the same number of stages (8 out of 12), so they will have the same latency for every processed packet. Difference between implementations in total of resources used is minuscule, making the overall resource usage near indistinguishable.

Throughput Both implementations were tested on a APS Advanced Programmable Switch BF2556X-1T, powered by a Barefoot BFN-T10-032D-020 Tofino 2.0T chip. The controller was programmed and results were collected using the P4Runtime API, running on the Ubuntu 20.04 operating system.

We test throughput by utilizing the packet generation feature included in the Tofino chip and checking if we are able to process it all at line-rate.

The compilation of both EPIC L0/L1 implementations was successful on the Tofino compiler, which means both solutions can run at terabit speeds, with enough servers (64) in our testbed (each server connected to one 100Gbps port). However, we do not have such a setup. As such, we utilized the packet generation feature (capable of around 50Gbps with small packets) in order to test our implementations. We found that the size of payload and hop fields leads to the same line rate result, which shows that indeed we achieve line rate.

B. EPIC L0/L1 against the state of the art

In this section, we compare our implementations of EPIC L0 and EPIC L1 with the state of the art. We will compare them mainly on two metrics: the security guarantees that each solution provides, and the performance, in terms of throughput, of each solution. We will show that our solution is capable of either matching or exceeding the packet processing performance of previous work, while delivering better security guarantees. Table III provides a high level overview of our results.

Security Both Soucková's [25] and Joeri and Schutijser's [22] solutions implement a version of the EPIC L0 dataplane protocol. The main difference between them is the type of cryptography operations. Soucková's NetFPGA solution performs the cryptographic MAC computations online, once per-packet. By contrast, Joeri and Schutijser's Tofino-based solution precomputes all valid MACs, stores them on a table and, in order to check the packet MAC's validity, it performs a table lookup to see if the MAC present on the packet is also present in the table.

Both our EPIC L0 and EPIC L1 implementations do on-demand cryptographic operations per packet. This is specially important for the EPIC L1 implementation: since this protocol includes a freshness guarantee for each packet, by adding a packet's timestamp to each packet's header, this is not compatible with Joeri and Schutijser's table-based approach, since it is not viable to do all precomputations ahead of time.

Performance We compare our results with the state of the art of Soucková's [25] and Joeri and Schutijser's [22] works. Soucková's implementation target was an NetFPGA, a device capable of a maximum throughput on all of its ports of 40Gbps (4 ports of 10Gbps). Their implementation was not able to fully fit onto this device, as explained in Section V-B. However, they were able to perform throughput measurements on a partial design, while accounting for different frame sizes (1500B and 115B frames). We compare our implementation measurements with these.

Joeri and Schutijser's [22] implementation was able to run at line-rate, while targeting a Tofino device, the same type of device as our solution. They performed measurements on a single 100Gbps port of the device, which is sufficient to show that the implementation runs at line-rate (a 128-port Tofino ASIC chip is capable of processing up to 12.8Tbps). We run similar measurements on our implementation as well, in order to allow better comparisons with this solution.

Our implementation is able to achieve the same speeds as Joeri and Schutijser's implementation and over two orders of magnitude faster than Soucková's theoretical maximum. However, like Soucková's implementation and unlike Joeri and Schutijser's, we do all of the cryptographic operations per-packet. This means our implementation of EPIC L1 is able to run at terabit speeds, while providing increased security guarantees, in the form of MAC freshness.

TABLE III
PROPERTIES OF EACH IMPLEMENTATION.
NOTE*: SINCE SOUCKOVÁ'S FULL IMPLEMENTATION WAS NOT CAPABLE OF FITTING IN THE TARGET, THIS IS THE ADVERTISED THEORETICAL MAXIMUM, TESTED ONLY ON THEIR PARTIAL DESIGN.

Implementation	Type of Cryptography implemented	Maximum throughput	Freshness
Soucková's [25]	On-demand	$\leq$ 40 Gbps*	No
Joeri and Schutijser's [22]	Table-based	12.8 Tbps	No
Our EPIC L0 implementation	On-demand	12.8 Tbps	No
Our EPIC L1 implementation	On-demand	12.8 Tbps	Yes

VIII. CONCLUSION AND FUTURE WORK

In this work, we made the case for the need of a new Internet architecture, that can overcome the security problems present today, while still being able to deliver the level of service we have come to expect. SCION is the first, new, production-ready Internet architecture, with a focus on security while still remaining efficient enough for today's use-cases. We have implemented a SCION-compatible router data plane in a programmable switch, powered by an Intel Tofino ASIC, with increased performance of over two orders of magnitude

TABLE II
RELATIVE RESOURCE USAGE DIFFERENCE BETWEEN EPIC L0 AND EPIC L1, IN EACH STAGE AND IN TOTAL. RESULTS TAKEN FROM BAREFOOT P4 INSIGHT TOOL. DIFFERENCES BETWEEN IMPLEMENTATIONS ARE ALMOST ZERO, MAKING THEM VIRTUALLY IDENTICAL. VARIATION IN SPECIFIC STAGES WHERE THE TOTAL BECOMES 0% CAN BE ATTRIBUTED BY SMALL, IRRELEVANT DEVIATIONS BY THE COMPILER.

Resource	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	Total
Action Data Bus Bytes	0.78%	0%	1.56%	2.34%	0%	0%	0.78%	0.78%	0%	0%	0%	0%	0.52%
Exact Match Input Xbar	-0.78%	-0.78%	0.78%	0.78%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Gateway	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Hash Bit	-2.40%	-2.40%	2.40%	2.40%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Logical Table ID	-6.25%	-6.25%	6.25%	6.25%	0%	0%	0%	0%	0%	0%	0%	0%	0%
SRAM	-1.25%	-1.25%	1.25%	1.25%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Stash	-6.25%	-6.25%	6.25%	6.25%	0%	0%	0%	0%	0%	0%	0%	0%	0%
VLIW Instruction	0%	-4.55%	4.55%	6.25%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Exact Match Search Bus	0%	-6.25%	6.25%	6.25%	0%	0%	0%	0%	0%	0%	0%	0%	0.52%
Exact Match Result Bus	-6.25%	-6.25%	6.25%	6.25%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Ternary Result Bus	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%

over the previous state of the art. We have implemented both EPIC L0 and EPIC L1. This last one provides more security guarantees than SCION, by being more resistant against brute force attacks. Both our implementation were written in P4, in a cross-platform manner, targeting any PISA-compatible device.

One of the major challenges in this work was the implementation of cryptographic primitives in our target, that were able to run at line-rate. In order to do this, we adapted the Simplified Even-Mansour cryptographic scheme and ported it to our target, since previous implementations of other cryptographic schemes were insufficient to get the throughput that we required. The permutation used on this implementation was based on a Substitution-Permutation-Substitution layer, where all substitution were performed by non-linear SBOXes, mapped to P4 tables for maximum efficiency. The permutation network was adapted from previous lightweight ciphers like GIFT-128 [2]. Overall, we were able to implement a full cryptographically-secure scheme in 8 stages of the device.

Finally, we compared our implementations with the previous state of the art solutions, showing that we were able to get comparable or faster throughput speeds, while providing improved security guarantees.

Our results show that both the EPIC family of protocols, and the SCION Internet architecture are viable solutions at high speeds, since they are suitable for hardware-backed implementations. As such, both can be considered for world wide large-scale deployments. We have also shown that programmable network devices, programmed with P4, can be use to implement relatively complex protocols while still performing at line-rate. This makes these devices extremely useful tools that can simplify the development and implementation of network protocols for a variety of different targets, while still providing enough performance for real-world cases.

Designing and implementing secure and efficient protocols, targeting these new programmable network devices, is a very promising area to explore. This allows us to solve long-standing architectural problems in our current infrastructure, while still being able to process great amounts of traffic, which is extremely important in maintaining the level of service we have come to expect from it. Some topics that could improve our work include:

- Implement the EPIC L2 and EPIC L3 protocols in modern programmable network devices: after implementing EPIC L0 and EPIC L1, the next logical step would be to implement the most secure siblings of this family of protocols. Further analysis can be made in order to find out how much each added security property contributes in the overall performance of each implementation.
- Design and implement better and safer cryptographic primitives in the data-plane: while the Simplified Even-Mansour scheme may be considered sufficient for various use cases, very little work has been done on developing cryptographic primitives for PISA devices. Further work can be done either by improving, both in efficiency and security, the current Simplified Even-Mansour primitive, or by creating a new cryptographic primitive, custom-made for this new architecture.

REFERENCES

[1] Changhoon Kim Anurag Agrawal. *Intel Tofino2 – A 12.9Tbps P4-Programmable Ethernet Switch*. Intel Corporation. Aug. 18, 2020. URL: https://hc32.hotchips.org/assets/program/conference/day2/HotChips2020_Networking_Tofino.pdf.

[2] Subhadeep Banik et al. “GIFT: a small present”. In: *International conference on cryptographic hardware and embedded systems*. Springer. 2017, pp. 321–345.

[3] Steven M Bellovin. “Security problems in the TCP/IP protocol suite”. In: *ACM SIGCOMM Computer Communication Review* 19.2 (1989), pp. 32–48.

[4] Pat Bosshart et al. “P4: Programming protocol-independent packet processors”. In: *ACM SIGCOMM Computer Communication Review* 44.3 (2014), pp. 87–95.

[5] Kevin Butler et al. “A survey of BGP security issues and solutions”. In: *Proceedings of the IEEE* 98.1 (2009), pp. 100–122.

[6] Xiaoqi Chen. “Implementing AES encryption on programmable switches via scrambled lookup tables”. In: *Proceedings of the Workshop on Secure Programmable Network Infrastructure*. 2020, pp. 8–14.

[7] Danny Cooper et al. “On the risk of misbehaving RPKI authorities”. In: *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*. 2013, pp. 1–7.

[8] Alex Davidson et al. “Tango or Square Dance? How Tightly Should we Integrate Network Functionality in Browsers?” In: *arXiv preprint arXiv:2210.04791* (2022).

[9] Orr Dunkelman, Nathan Keller, and Adi Shamir. “Minimalism in cryptography: The Even-Mansour scheme revisited”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2012, pp. 336–354.

[10] Tim Griffin and Geoff Huston. *BGP wedgies*. RFC 4264. IETF, 2005. URL: https://datatracker.ietf.org/doc/html/rfc4264.

[11] Timothy G Griffin, F Bruce Shepherd, and Gordon Wilfong. “The stable paths problem and interdomain routing”. In: *IEEE/ACM Transactions On Networking* 10.2 (2002), pp. 232–243.

[12] Nate Kushman, Srikanth Kandula, and Dina Katabi. “Can you hear me now?! it must be BGP”. In: *ACM SIGCOMM Computer Communication Review* 37.2 (2007), pp. 75–84.

[13] Markus Legner et al. “EPIC: Every Packet Is Checked in the Data Plane of a Path-Aware Internet”. In: *29th USENIX Security Symposium (USENIX Security 20)*. 2020, pp. 541–558.

[14] Qi Li, Yih-Chun Hu, and Xinwen Zhang. “Even rockets cannot make pigs fly sustainably: Can BGP be secured with BGPsec”. In: *Workshop SENT'14, 23 February 2014, San Diego, USA, Copyright 2014 Internet Society: Proceedings*. Internet Society. 2014.

[15] Qi Li et al. “BGP with BGPsec: Attacks and countermeasures”. In: *IEEE Network* 33.4 (2018), pp. 194–200.

[16] Robert Lychev, Sharon Goldberg, and Michael Schapira. “BGP security in partial deployment”. In: *Is the juice worth the squeeze* (2013).

[17] K. Sriram M. Lepinski. *BGPsec Protocol Specification*. RFC 8205. IETF, Sept. 2017. URL: https://datatracker.ietf.org/doc/html/rfc8205.

[18] Adrian Perrig et al. *SCION: a secure Internet architecture*. Springer, 2017.

[19] Jon Postel. *INTERNET PROTOCOL*. RFC 791. IETF, Sept. 1981. URL: https://datatracker.ietf.org/doc/html/rfc791.

[20] R. Austein R. Bush. *The Resource Public Key Infrastructure (RPKI) to Router Protocol*. RFC 6810. IETF, Jan. 2013. URL: https://datatracker.ietf.org/doc/html/rfc6810.

[21] NCC Ripe. “Youtube hijacking a ripe ncc ris case study”. In: *http://www. ripe. net/news/study-youtube-hijacking. html* (2008).

[22] Joeri de Ruiter and Caspar Schutijser. “Next-generation internet at terabit speed: SCION in P4”. In: *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*. 2021, pp. 119–125.

[23] *SCION Header Specification*. Anapaya Systems. 2021.

[24] SCIONLab. *SCION*. https://github.com/scionproto/scion.

[25] Kamila Soucková. “FPGA-based line-rate packet forwarding for the SCION future Internet architecture”. MA thesis. ETH Zurich, 2019.

[26] Daniel Wagner et al. “United We Stand: Collaborative Detection and Mitigation of Amplification DDoS Attacks at Scale”. In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 970–987.

[27] T. Li Y. Rekhter. *A Border Gateway Protocol 4 (BGP-4)*. RFC 1654. IETF, July 1994. URL: https://datatracker.ietf.org/doc/html/rfc1654.

[28] Noa Zilberman et al. “NetFPGA SUME: Toward 100 Gbps as research commodity”. In: *IEEE micro* 34.5 (2014), pp. 32–41.