Chương trình KC-01:
Nghiên cứu khoa học
phát triển công nghệ thông tin
và truyền thông

Đề tài KC-01-01:
Nghiên cứu một số vấn đề bảo mật và
an toàn thông tin cho các mạng dùng
giao thức liên mạng máy tính IP

Báo cáo kết quả nghiên cứu

GIỚI THIỆU MỘT SỐ KẾT QUẢ MỚI TRONG BẢO MẬT MẠNG DÙNG GIAO THỨC IP, AN TOÀN MẠNG VÀ THƯƠNG MẠI ĐIỆN TỬ

Quyển 1C: "TÌM HIỂU KHẢ NĂNG CÔNG NGHỆ ĐỂ CỨNG HOÁ CÁC THUẬT TOÁN MẬT MÃ"

HÀ NỘI-2004

Báo cáo kết quả nghiên cứu

GIỚI THIỆU MỘT SỐ KẾT QUẢ MỚI TRONG BẢO MẬT MẠNG DÙNG GIAO THỨC IP, AN TOÀN MẠNG VÀ THƯƠNG MẠI ĐIỆN TỬ

Quyển 1C: "TÌM HIỂU KHẢ NĂNG CÔNG NGHỆ ĐỂ CỨNG HOÁ CÁC THUẬT TOÁN MẬT MÃ"

Chủ trì nhóm nghiên cứu
TS. Nguyễn Hồng Quang

Mục lục

MỞ ĐẦU

Mật mã có thể thực hiện theo cách thủ công hoặc tự động với sự trợ giúp của máy móc. Mật mã thủ công hầu như chỉ được nhắc đến như một nhân tố trong lịch sử. Những nhược điểm của mật mã thủ công bao gồm độ phức tạp của thuật toán thấp, tốc độ chậm, chỉ bảo mật được với một số loại nguồn tin, mức độ sai sót và tính an toàn phụ thuộc nhiều vào con người...

Trong thời đại điện tử, truyền thông và tin học ngày nay *các nguồn tin ngày càng đa dạng*; mọi *thông tin đều được số hóa* với khổng lồ trữ lượng tại chỗ và lưu lượng trên kênh; *đòi hỏi của người dùng ngày càng cao* về độ mật, tốc độ, độ an toàn, tính tiện dụng... Trong tình hình đó, chỉ có một lựa chọn duy nhất là thực hiện mật mã với sự trợ giúp của máy móc.

Thuật ngữ máy móc nói đến ở đây không bao gồm tất cả mọi loại hình kỹ thuật (cơ khí, cơ điện...), mà ám chỉ trong phạm vi hẹp là các thiết bị điện tử bởi điện tử là ngành thích hợp nhất để thỏa mãn các yêu cầu về xử lý tín hiệu số, thuật toán phức tạp và dễ update, tốc độ cao, kích thước nhỏ, giá thành hạ...

Khi điện tử hóa các bài toán mật mã thường bắt gặp hai câu hỏi sau.

Câu hỏi thứ nhất, là nên thực hiện mật mã trên cơ sở phần cứng (hardware) hay phần mềm (software)?. Để trả lời cho câu hỏi đó cần *phân tích các ưu nhược điểm* của hai platform này, xác định *những yêu cầu chung* cho một thiết bị điện tử và *yêu cầu riêng* mang tính đặc thù của thiết bị mật mã, các *yếu tố cần cân nhắc* khi sử dụng thực tế.

Câu hỏi thứ hai là, công nghệ nào thích hợp với mật mã? Không

như ở lĩnh vực khác chỉ cần chọn đúng công nghệ để thực hiện bài toán đặt ra sao cho tối ưu về giá thành, dễ phát triển, nhanh ra thị trường, có khả năng upgrade... là đủ. Với ngành mật mã, ngoài việc chọn công nghệ thích hợp cho *encryption*, cũng quan trọng không kém là công nghệ đó có bảo đảm *security* không.

Để tiến đến mục tiêu "*Tìm hiểu khả năng công nghệ, chuẩn bị kiến thức để cứng hóa các thuật toán mật mã*", cần thiết phải nghiên cứu trả lời hai câu hỏi trên một cách toàn diện. Như vậy tài liệu này gồm 3 phần chính sau:

Phần 1: So sánh thực hiện mật mã bằng phần cứng và phần mềm

Phần 2: Lựa chọn công nghệ cho cứng hóa mật mã

Phần 3: Chuẩn bị để cứng hóa bằng FPGA

PHẦN 1
SO SÁNH THỰC HIỆN MẬT MÃ BẰNG PHẦN CỨNG VÀ PHẦN MỀM

Mục tiêu phần này là trả lời câu hỏi: nên thực hiện mật mã bằng phần cứng hay phần mềm; khi nào nên chọn phần cứng, khi nào nên chọn phần mềm, và khi nào nên phối hợp cả hai.

1.1 CÁC PLATFORM HARDWARE, SOFTWARE VÀ FIRMWARE

Định nghĩa Hardware, Software và Firmware [1] như sau:

- *Hardware:* thiết bị vật lý để xử lý các chương trình và số liệu
- *Software:* các chương trình và dữ liệu liên quan có thể được viết và thay đổi động.
- *Firmware:* các chương trình và số liệu (tức là software) được lưu trữ vĩnh viễn trong phần cứng (chẳng hạn trong ROM, PROM, or EPROM) sao cho chúng không thể được viết và thay đổi động trong khi thực hiện. Các chương trình và số liệu lưu trong EEPROM được xem là software[1].

Các thuật toán chung và các thuật toán mật mã nói riêng có thể được thực hiện trên *hardware, software hay firmware*. Trong đó hardware bao gồm cả các linh kiện có chức năng cố định (các ICs logic) lẫn các linh kiện có chức năng lập trình cứng được (PLD, ASIC, FPGA); software bao gồm cả các phần mềm chạy trên máy tính PC lẫn các phần mềm chạy trên các vi xử lý chuyên dụng. Sự phân chia này đôi khi không thật rành

[1] Bởi vì ROM, PROM, or EPROM là các linh kiện mà muốn xóa hay thay đổi nội dung của nó phải cần thiết bị chuyên dụng; còn EEPROM thì có thể lập trình để xóa hay thay đổi nội dung.

mạch bởi tùy theo vi mạch sử dụng (ví dụ dùng EPROM hay EEPROM) mà cùng một thiết kế nhưng được xem là nghiêng về phía hardware hay software. Có thể coi khái niệm *firmware* là phần sụn, là trung gian giữa phần cứng và phần mềm. Và tùy thuộc vào ngữ cảnh mà firmware được xem là cứng hay mềm nên trong tài liệu này chúng ta coi là có hai platform chính là hardwae và software.

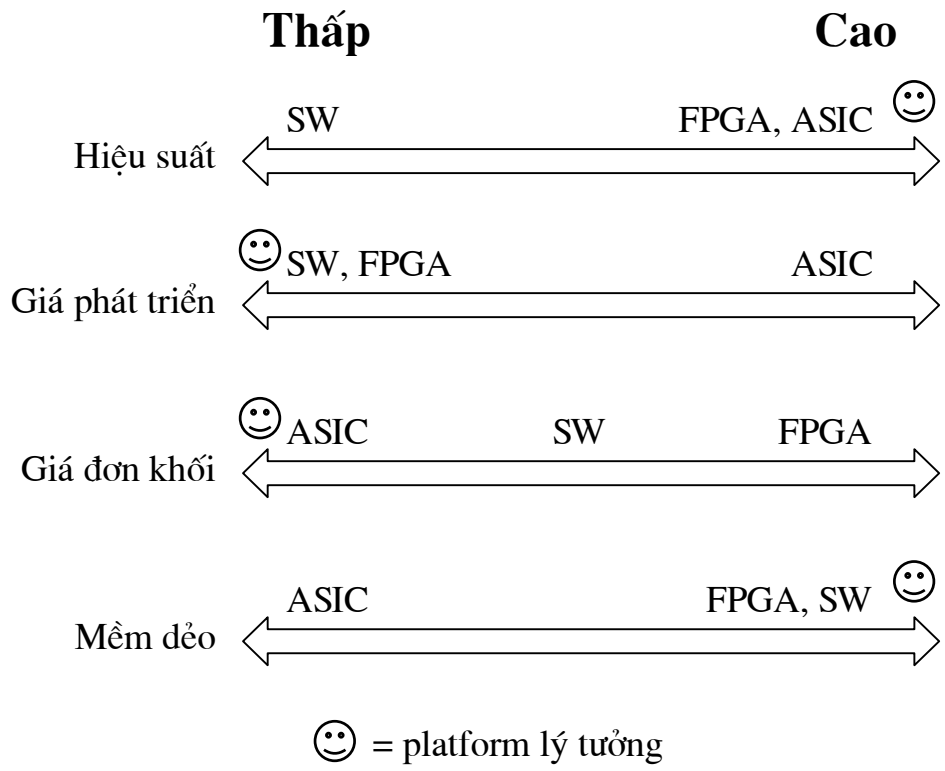
Cũng cần chú thích thêm là các khái niệm trên chỉ mang tính cục bộ. Trong một thiết bị điện tử có thể có nhiều khối, mỗi khối có thể dựa trên platform hardware, software hay firmware. Ngày nay rất ít thiết bị nào thiết kế chỉ dựa trên một platform nào đó.

1.2 CHỌN PLATFORM NÀO ĐỐI VỚI THIẾT KẾ NÓI CHUNG

Trước khi bắt tay vào thiết kế cần thiết phải xác định platform sẽ sử dụng. Việc lựa chọn theo các tiêu chuẩn sau [2], [3]:

- Thời gian đưa sản phẩm ra thị trường
- Tốc độ thực hiện thuật toán
- Giá thành, bao gồm
  - Giá đơn khối: tức giá thành sản phẩm
  - Giá phát triển: tức giá nghiên cứu, thiết kế chế tạo
- Năng lượng tiêu thụ: chú trọng với thiết bị di chuyển và không dây
- Tính mềm dẻo: để dễ dàng thay đổi tham số, thuật toán, cấu hình

Chọn platform tùy thuộc vào việc coi trọng tiêu chuẩn nào. Hình 1 so sánh giữa hardware, software và FPGA [3]. So sánh cho thấy, ở một mức độ nào đó thì FPGA là tổng hợp các ưu điểm của hardware và software.



Hình 1. So sánh hardware, software và FPGA

1.3 CHỌN PLATFORM NÀO ĐỐI VỚI THIẾT KẾ MẬT MÃ

Đối với mật mã cần quan tâm đến hai vấn đề chính: mật mã và an toàn mật mã. Do đó ngoài những tiêu chí để lựa chọn chung như trên cần quan tâm đến các yêu cầu sau:

- Độ mềm dẻo mật mã: khả năng thay đổi tham số, khóa, thuật toán
- Tốc độ mã hóa nhanh
- Độ an toàn vật lý: chống truy nhập trái phép

Yêu cầu thứ nhất gần giống như đối với các thiết bị điện tử nói chung. Điểm khác là trong các thiết bị điện tử nói chung, yêu cầu này để upgrade chức năng trong tương lai; còn trong thiết bị mật mã yêu cầu thay đổi này liên tục được sử dụng, ví dụ đổi cấu hình khi chuyển liên lạc sang

mạng khác, hoặc thay đổi khóa và thuật toán trong mỗi phiên liên lạc.

Tốc độ mã hóa là yếu tố quan trọng đặc biệt trên các luồng tốc độ cao với thông lượng dữ liệu lớn. Tốc độ mã hóa đủ lớn sẽ làm cho cảm giác mật mã trở nên "trong suốt" và người dùng dễ chấp nhận mật mã hơn.

Yêu cầu an toàn có thể thực hiện bằng nhiều hình thức khác nhau tùy theo mức độ yêu cầu: đó có thể là các quy định, hay mật khẩu, hay hình thức xác thực vai trò, như các mức 1 và 2 trong [1]. Các biện pháp an toàn vật lý bằng phần cứng hiệu quả hơn các biện pháp bằng phần mềm hay bằng quy định [4]. Đó có thể là các khóa cơ khí, dấu niêm phong, các mạch điện tử phát hiện và hủy số liệu khi có xâm nhập trái phép, như các mức 3 và 4 trong [1]. Tuy nhiên cũng phải thấy là các biện pháp an toàn vật lý bằng phần cứng bao giờ cũng đắt tiền hơn phần mềm.

Nhưng platform nào, hardware hay software, là phù hợp với mật mã? Khi xét đến các yêu cầu chung và riêng ta có câu trả lời là cả hai. Các yêu cầu cụ thể để xem xét có thể như Bảng 1.1 và Bảng 1.2 sau:

Bảng 1.1 So sánh giữa hardware và software

Yêu cầu thực tế	Hardware	Software
Độ an toàn	×	
Tốc độ	×	
Tính mềm dẻo	×	×
Giá phát triển		×
Giá thành phẩm		×
Năng lượng tiêu thụ	×	

Bảng 1.2 So sánh ASIC, FPGA và software về các đặc điểm dùng cho mật mã

	ASIC	FPGA	*Software*

Xử lý song song	Có	Có	*Giới hạn*
Pipelining	Có	Có	*Giới hạn*
Kích thước từ	Có thể thay đổi	Có thể thay đổi	*Cố định*
Tốc độ	Rất nhanh	Nhanh	*Nhanh vừa phải*
Linh hoạt về thuật toán	Không	Có	*Có*
Chống xâm nhập	Mạnh	Giới hạn	*Yếu*
Điều khiển khóa	Mạnh	Vừa phải	*Yếu*
Thời gian thiết kế	Dài	Dài vừa phải	*Ngắn*
Công cụ thiết kế	Rất đắt	Đắt vừa phải	*Không đắt*
Testing	Rất đắt	Đắt vừa phải	*Không đắt*
Bảo trì và nâng cấp	*Đắt*	*Không đắt*	*Không đắt*

Bảng 1.2 là so sánh về các đặc điểm dùng cho mật mã của giải pháp hardware, mà đại diện là ASIC và FPGA, và giải pháp software, mà đại diện là các bộ xử lý mục đích chung [20].

1.4 SO SÁNH VỀ ĐỘ AN TOÀN

NSA (National Security Agency) nói rằng chỉ thực hiện bằng hardware mới thực sự được coi là an toàn [5]. Mặc dù vậy trong thực tế người dùng thường thích các giải pháp bằng software hơn, có lẽ do tính tiện dụng và giá thành của nó. Tuy nhiên ở cấp độ chính phủ và an ninh quốc phòng nơi đối tượng bảo mật là các thông tin nhạy cảm cấp quốc gia thì tính an toàn cho các thiết bị mật mã cần phải được nhấn mạnh.

Trong [1] đã phân độ an toàn thành 4 mức, trong đó mức càng cao càng phải sử dụng nhiều phần cứng vật lý. So với các giải pháp an toàn bằng hardware, giải pháp bằng software có những nhược điểm sau [4]:

- Sử dụng chung không gian bộ nhớ với các ứng dụng khác,
- Chạy trên đỉnh hệ điều hành
- Rất dễ bị modify.

Chúng ta sẽ phân tích kỹ hơn về điều này.

1.4.1 SỬ DỤNG CHUNG KHÔNG GIAN NHỚ RAM

Thường thì giải pháp software phải sử dụng RAM thông qua các dịch vụ của hệ điều hành. RAM cũng có thể được xâm nhập bởi software khác. Mặc dù hầu hết hệ điều hành đều có cách bảo vệ RAM nhưng việc bảo vệ này chỉ để tăng sức khỏe hệ điều hành chứ không nhằm mục đích security. Thứ hai, đối với bộ nhớ thứ cấp, việc bảo vệ khó hơn và yếu hơn nhiều.

RAM của các module mật mã cần được bảo vệ đặc biệt. Hầu hết các thuật toán mật mã và giao thức cần lưu trữ kết quả trung gian trong khi module làm việc. Các kết quả trung gian này chính là các giá trị có thể liên quan rất mật thiết với khóa mật (thậm chí chính là khóa). ***Do đó mức độ an toàn của các module mật mã software bị giới hạn bởi mức độ an toàn của cơ chế bảo vệ tính bí mật và tính toàn vẹn của không gian nhớ nó sử dụng***. Điều này thường không được đánh giá một cách thích đáng. Nếu các kết quả trung gian này bị rò rỉ thì toàn hệ thống có thể dễ dàng bị xâm hại.

Bộ nhớ thứ cấp thường yêu cầu cùng mức độ bảo vệ mật như bộ nhớ sơ cấp. Thường nó được dùng để lưu trữ chính chương trình, khóa dài hạn và số liệu. Tuy nhiên việc giữ cho bộ nhớ thứ cấp được bí mật trên nền các ứng dụng được chia xẻ đang rất bị xem nhẹ. Việc bảo vệ bộ nhớ thứ cấp thực tế thường chỉ bằng cách mã hóa. Nhưng, việc mã hóa lại tăng độ phức tạp của vấn đề lưu trữ khóa mã.

Trong giải pháp hardware do không gian nhớ trong được quản lý riêng nên giải quyết được vấn đề bảo vệ bộ nhớ. Thêm nữa giải pháp hardware có thể được áp dụng bằng các biện pháp hardware để ngăn ngừa xâm nhập trái phép bộ nhớ, điều đó tự nhiên an toàn hơn nhiều các dịch vụ của hệ điều hành mà software chạy trên nó.

1.4.2 BẢO ĐẢM TOÀN VẸN

Phần mềm là một tập các lệnh trong bộ nhớ. Do việc bảo vệ bộ nhớ thứ cấp không bảo đảm nên tính toàn vẹn của mã lệnh cũng không được bảo đảm. Đối phương có thể thay đổi code của ứng dụng hoặc làm rò rỉ nghiêm trọng thông tin. Việc thay đổi có thể kiểu nhân công, hoặc tự động bằng chương trình kiểu virus hay con ngựa Trojan.

Giải pháp hardware an toàn ở chỗ các mã lệnh đã được đốt trong IC. ***Đốt vật lý mã nguồn là cách tốt nhất để không thể modify nó. Đây là cách mà bất kỳ module mật mã nào cũng nên làm.***

1.4.3 THÁM NGƯỢC THIẾT KẾ

Giải pháp software thường dễ bị đối phương đọc và rất dễ thám ngược thiết kế. Do software chỉ là các lệnh trong bộ nhớ, mà bộ nhớ thường không được bảo vệ nên đối phương dễ dàng đọc mã nguồn và suy ra thuật toán với một chi phí nào đó.

Đối với hardware, cấu trúc mạch hay mã nguồn đều được đốt vật lý nên không thể xem do đó cũng không thể thám ngược thiết kế[2].

1.4.4 TẤN CÔNG PHÂN TÍCH NĂNG LƯỢNG

Giải pháp software rất dễ bị tổn thương với tấn công dựa trên phân tích năng lượng tiêu thụ. Mỗi lệnh software được compiler dịch thành tập các lệnh ngôn ngữ máy. Các mã máy này có mẫu tiêu thụ năng lượng đã xác định. Các mẫu này rất dễ nhận dạng bằng các kỹ thuật phân tích năng lượng tương đối đơn giản. Nhờ đó có thể thu thập thông tin về trạng thái bên trong của module và thám ra khóa mật.

Giải pháp hardware có thể áp dụng các biện pháp đặc biệt che dấu sự thăng giáng của tiêu thụ năng lượng, ngăn cản kẻ tấn công thu thập thông tin về tiêu thụ năng lượng nhằm thám khóa mật.

---

[2] Thực ra giải pháp nào thì về nguyên tắc cũng đều có thể thám được. Nhưng với hardware chi phí cho nó có thể rất lớn so với software và thời gian cũng dài hơn.

1.4.5 VẤN ĐỀ LƯU TRỮ KHÓA DÀI HẠN

Bài toán lưu trữ khóa có thể xem là một phần của bài toán bảo vệ bộ nhớ đã nói phần trước. Việc lưu trữ khóa dài hạn yêu cầu sử dụng bộ nhớ thứ cấp và đây là cơ hội cho các tấn công.

Khóa dài hạn phải được lưu trữ sao cho bảo vệ được tính mật và tính toàn vẹn của nó. Thêm nữa do khóa này là dài hạn (đối lập với khóa phiên) chúng phải được lưu trữ trong bộ nhớ bất biến. Do loại bộ nhớ này thường có thể đọc được bởi thiết bị chuyên dùng nên khóa mã dùng để bảo vệ khóa dài hạn cũng phải bí mật và toàn vẹn.

Có hai giải pháp chung để cất giữ khóa-mã-khóa. Giải pháp thứ nhất là lấy từ mật khẩu người dùng và không cần cất giữ. Khóa được tạo ra khi người dùng nhập mật khẩu vào. Nếu người dùng nhập đúng mật khẩu thì khóa được tạo ra một cách bình thường và sẽ được dùng để giải mã khóa dài hạn. Có một số nhược điểm với kỹ thuật này, hai trong số đó là: 1) kỹ thuật này không thể áp dụng cho hệ thống tự động không người vận hành, tức là không có ai để nhập mật khẩu. 2) Do entropy (tính bất định?) của mật khẩu thấp, vì mật khẩu không được dùng lâu và có thể đoán được.

Giải pháp thứ hai là mã hóa khóa dài hạn bằng khóa trong được cất giữ ở đâu đó trong ứng dụng. Như thế là ***đặt độ an toàn dựa trên khả năng che dấu khóa trong***. Khi sử dụng giải pháp này, software ***thường yếu vì không thể có một vị trí ẩn đủ tốt để che dấu khóa***. Do software nằm trên không gian nhớ có thể xâm nhập[3], và do việc thám ngược thiết kế software nhiều khả thi nên các khóa ẩn trong software thường được thám ra với một mức độ đầu tư nào đó.

[3] Nói chung độ an toàn của mã nguồn không thể bảo đảm. Việc đặt độ an toàn của hệ thống trên độ an toàn thực hiện của chính nó được gọi là "*Độ an toàn mờ mịt*" và trong thực tế được coi là không an toàn.

Với hardware, có các giải pháp hiệu quả hơn để che dấu khóa. Các khóa trong có thể được đốt như một phần của phần cứng do đó cực kỳ khó thám chúng. Khóa trong cũng có thể được cất trên bộ nhớ bất biến mà các ứng dụng khác không thể thâm nhập được do cách thiết kế hardware.

1.4.6 PHỤ THUỘC VÀO ĐỘ AN TOÀN CỦA HỆ ĐIỀU HÀNH

Khi một ứng dụng chạy trên đỉnh của ứng dụng khác lớp thấp hơn (hệ điều hành chẳng hạn) thì độ an toàn của ứng dụng lớp cao hơn phụ thuộc nhiều điểm vào độ an toàn của ứng dụng mức thấp hơn ở khía cạnh lỗi. Xảy ra như sau, nếu lỗi sai xảy ra trong hoạt động của hệ điều hành thì lỗi này dẫn đến thêm khả năng tổn thương của ứng dụng chạy trên đỉnh của nó. Nói chung mỗi vấn đề an toàn của hệ điều hành, hoặc đã biết hoặc còn chưa biết, đều có thể gây ra các vấn đề an toàn với module mật mã. Một ví dụ điển hình cho hiện tượng này là các hệ điều hành để rò rỉ nội dung bộ nhớ qua các file tráo đổi (swap files) và lỗi trong quản lý bộ nhớ và sơ đồ bảo vệ của hệ điều hành. Các hệ điều hành mở hoặc các hệ điều hành cung cấp các dịch vụ mức cao thậm chí còn có nhiều vấn đề hơn. Các mức dịch vụ của hệ điều hành càng cao thì tiềm ẩn loại lỗi này càng lớn.

Phần cứng không phụ thuộc vào các dịch vụ của hệ điều hành mức cao và do đó không phụ thuộc vào tính an toàn của các dịch vụ này.

Tóm lại, trong phần này chúng ta đã xem xét một số vấn đề chính yếu nhất nhằm trả lời câu hỏi: nên thực hiện mật mã bằng phần cứng hay phần mềm. Câu trả lời là ***cả hai,*** nhưng tùy vào yêu cầu thực tế:

- Đối với *yêu cầu độ an toàn cao, tốc độ lớn* → nên chọn platform là ***hardware***.
- Đối với độ an toàn thấp, tốc độ thấp, nhưng *yêu cầu rẻ* → nên chọn

platform là ***software***.

PHẦN 2
LỰA CHỌN CÔNG NGHỆ CHO CỨNG HÓA MẬT MÃ

Nội dung phần này là tìm hiểu các công nghệ hiện có, chọn một công nghệ thích hợp để cứng hóa mật mã, phân tích an toàn mật mã với hardware nói chung và công nghệ lựa chọn nói riêng. Với mục tiêu xác định này chúng ta sẽ chỉ bàn luận về hardware.

Giả thiết yêu cầu đặt ra là bảo mật thông tin trong khu vực Chính phủ, An ninh và Quốc phòng ở đó đòi hỏi độ an toàn cao và tốc độ lớn, rõ ràng platform lựa chọn phải là hardware. Tuy nhiên trong thế giới hardware có nhiều công nghệ khác nhau. Vậy câu hỏi tiếp theo sẽ là: *Chọn công nghệ nào là phù hợp cho mật mã?*

Chúng ta sẽ bắt đầu với việc phân tích 7 công nghệ xử lý tín hiệu trong thời gian thực phổ biến nhất hiện nay. Từ đó rút ra kết luận cần thiết.

Cũng cần chú thích là trong số 7 công nghệ sẽ phân tích, nhiều công nghệ là sự pha trộn giữa hardware và software trên cơ sở lập trình cho chip. Tuy nhiên khác với software như đã đề cập ở phần trước ở chỗ software cho chip thực hiện trên hardware được thiết kế riêng, chuyên dụng, đóng kín, không dùng chung bộ nhớ và hệ điều hành, được đốt vật lý trên chip. Và như vậy có thể xếp chúng vào hardware platform.

2.1 PHÂN TÍCH CÁC CÔNG NGHỆ HIỆN NAY

Ngày nay có vô số công nghệ mà các nhà thiết kế điện tử phải đối mặt. Các công nghệ đều nhằm lợi ích người dùng là thiết bị phải nhỏ hơn,

nhanh hơn, thông minh hơn, tiêu thụ ít năng lượng hơn, tương tác được với nhau... nhưng cũng làm các nhà thiết kế bối rối nhiều hơn khi lựa chọn công nghệ thích hợp cho sản phẩm của mình. Theo hãng Taxas [6] thì có 7 công nghệ phổ biến nhất hiện nay cho bài toán xử lý tín hiệu trong thời gian thực, là ASIC, ASSP, vi xử lý có thể cấu hình, DSP, FPGA, MCU và RISC/GPP. Tiêu chí để đánh giá so sánh chúng bao gồm:

- Thời gian đưa sản phẩm ra thị trường ★★★
- Năng lực thực hiện ★★★
- Giá thành ★★★
- Dễ phát triển ★★★
- Năng lượng tiêu thụ ★★
- Tính mềm dẻo ★

(Các ngôi sao (★) xác định tầm quan trọng của các tiêu chí)

Thời gian đưa sản phẩm ra thị trường: đây là tiêu chí quan trọng nhất trong một chu trình phát triển, từ vài năm đến vài tháng. Theo bài báo "Mind of the Engineer" của Cahners thì thậm chí *Thời gian ra thị trường* còn điều khiển cả nền công nghiệp.

Năng lực thực hiện: là tiêu chí quyết định năng lực của sản phẩm. Tăng năng lực thực hiện sẽ thêm chức năng và tốc độ cao hơn, cũng như giảm kích thước và đạt chất lượng cao hơn.

Năng lực thực hiện có thể đo bằng nhiều cách, nói chung là số triệu thao tác trên một giây (MIPS), số triệu phép nhân trên một giây (MMACS); hoặc, đôi khi, đơn giản hơn đo bằng số chu kỳ clock trên một giây (MHz).

Giá thành: thường là tiêu chí hiển nhiên nhất, nhưng trong đa số trường hợp vẫn xếp sau *Thời gian ra thị trường* và *Năng lực thực hiện*. Nói chung số lượng sản phẩm và khách hàng càng nhiều thì tiêu chí này càng được đẩy lên cao và giá càng thấp.

Dễ phát triển: tiêu chí này đi cùng với hỗ trợ phát triển, công cụ phát triển, giá phát triển và có thể phân chi tiết thành *Hỗ trợ kỹ thuật, Đào tạo kỹ thuật, Trang web có giá trị của thành phần thứ ba, Công cụ phần mềm, Tài liệu, Thời gian thiết kế.*

Rõ ràng là càng nhiều hỗ trợ kỹ thuật thì người kỹ sư thiết kế càng có điều kiện tập trung vào công việc sáng chế của mình, thay vì phải tự nghiên cứu thì anh ta có thể thuê ý kiến của các chuyên gia.

Công cụ phát triển cũng là chìa khóa để thiết kế. Các công cụ mạnh như *DSP Starter Kits, Môi trường phát triển tích hợp, Compiler* và *Công cụ cho phần cứng đích...* giúp thiết kế trực quan và dễ dàng hơn.

Tát cả những điều đó đều cho phép rút ngắn đáng kể thời gian phát triển và thời gian ra thị trường, cũng đồng nghĩa với giảm tổng chi phí phát triển và hạ giá thành sản phẩm.

Năng lượng tiêu thụ: Tiêu chí này quan trọng trong các sản phẩm xách tay như điện thoại di động... Năng lượng tiêu thụ thấp tức là thời gian sống của ắcquy kéo dài – mỗi quan tâm lớn của người dùng cuối. Năng lượng tiêu thụ thấp cũng làm giảm phát xạ nhiệt, điều này có ý nghĩa lớn trong các thiết bị kín vì nhiệt độ trong vỏ kín tăng cao sẽ là nhân tố giảm mật độ kênh hoặc một số chức năng của thiết bị.

Tính mềm dẻo: là khả năng sửa đổi hay tăng thêm chức năng cho thiết bị khi có yêu cầu. Chẳng hạn các thiết bị làm việc theo chuẩn (như chuẩn truyền thông hay nén) được tung ra thị trường trong khi chuẩn còn

đang tạm thời. Như thế các nhà thiết kế cần tính toán sao cho sản phẩm có khả năng upgrade một cách dễ dàng và nhanh chóng khi chuẩn đã được phê chuẩn.

Sau đây chúng ta sẽ phân tích 7 công nghệ phổ biến nhất với từng tiêu chí kể trên.

2.1.1 CÔNG NGHỆ ASIC

ASIC (Application-Specific Integrated Circuit): Mạch tính hợp cho ứng dụng xác định.

Thời gian đưa sản phẩm ra thị trường của ASIC được coi là kém. Thời gian để thực hiện và test một sản phẩm ASIC và bộ xử lý có thể cấu hình có thể kéo dài từ hàng tháng đến hàng năm.

Năng lực thực hiện của ASIC được coi là tuyệt vời. Người thiết kế có thể thiết kế ASIC và FPGA sâu ở mức cổng để sản phẩm đạt hiệu suất sử dụng tài nguyên cao, sát với yêu cầu ứng dụng.

Giá thành của ASIC được coi là tuyệt vời. Thiết kế đến từng cổng logic cho phép kích thước vi mạch hiệu quả nhất và nhỏ nhất và cũng cho phép tính giá thành trên từng cổng.

Năng lượng tiêu thụ của ASIC được coi là tốt nếu chủ động thiết kế nhằm vào mục tiêu này. Các bộ xử lý cũng có hiệu quả tương tự. Tuy nhiên hầu hết các thiết kế trên ASIC lại nhắm vào hiệu suất thực hiện và giá thành chứ không phải vì năng lượng tiêu thụ.

Tiêu chí *Dễ phát triển* ASIC bị coi là khá. Mặc dù ASIC có thể coi là không đắt nhưng thực tế nếu tính cả chi phí cho phát triển thì ASIC lại là đắt nhất. Về công cụ phát triển, các nhà cung cấp ASIC chỉ hỗ trợ chung chứ không có cho riêng một ứng dụng xác định nào do kiến thức về

ASIC ở họ cũng yếu.

Tính mềm dẻo của ASIC bị coi là kém. Một thiết kế đã thực hiện trên ASIC thì không thể thay đổi hay bổi sung thêm gì trừ khi làm một thế hệ mới.

2.1.2 CÔNG NGHỆ ASSP

ASSP (Application-Specific Standard Product): Sản phẩm chuẩn cho ứng dụng xác định.

Thời gian ra thị trường của ASSP: nếu thị trường đã có và sản phẩm cũng đã sẵn sàng thì tiêu chí này từ tốt cho đến rất tốt. Nếu thị trường mới và còn phải phát triển các đặc điểm mới cho sản phẩm thì tiêu chí này là kém. Thỏa hiệp của cả hai khả năng ấy thì tiêu chí *Thời gian ra thị trường* của ASSP được coi là khá.

Giá thành của ASSP được coi là tốt cho loạt sản phẩm nhỏ, tuy nhiên kém hơn chút ít so với DSP.

Năng lượng tiêu thụ của ASSP được coi là rất tốt khi nó được thiết kế tối ưu cho ứng dụng xác định. Tuy nhiên nếu thay vì *Năng lượng tiêu thụ* mà thiết kế hướng đến *Giá thành* thì tiêu chí này thua DSP.

Dễ phát triển của ASSP được coi là khá, vì giả thiết một số khó khăn khi thiết kế các đặc điểm riêng biệt làm chậm quá trình phát triển. Tài liệu phát triển chung không tốt vì ASSP định hướng cho thiết kế chuyên dụng.

Tính mềm dẻo của ASSP bị coi là kém vì ngay từ đầu ASSP đã định hướng cho các sản phẩm đích xác định.

2.1.3 CÔNG NGHỆ CONFIGURABLE PROCESSOR

Configurable Processor: Bộ xử lý có khả năng cấu hình.

Thời gian ra thị trường bị coi là kém, nhưng *Năng lực thực hiện* lại được đánh giá là rất tốt do có thể cấu hình đặc biệt cho ứng dụng xác định.

Về *Giá thành* và *Năng lượng tiêu thụ* được coi là tốt. Về tính *Dễ phát triển* thì kém.

Tính mềm dẻo được đánh giá là khá: có thể thay đổi cấu hình để có được đặc điểm mới, tuy nhiên do định hướng cho ứng dụng xác định nên sau khi đã đưa ra vào sử dụng thì tính mềm dẻo trở nên kém.

2.1.4 CÔNG NGHỆ DSP

DSP (Digital Signal Processor): Bộ xử lý tín hiệu số.

Thời gian ra thị trường của DSP được coi là rất tốt. Các bộ xử lý có thể lập trình như DSP, RISC và MCU đều có khả năng lập trình bằng phần mềm để có được các chức năng và đặc điểm khác nhau, tiết kiệm thời gian so với các thực hiện tương tự bằng phần cứng. Trong ba loại kể trên thì thì DSP được coi là tốt nhất và cũng vì thế mà công cụ và thông tin dành cho DSP cũng nhiều nhất.

Năng lực thực hiện của DSP được coi là rất tốt. Các DSP có cấu trúc multi-MAC VLIW như TMS320C6000 có tốc độ MIPS rất cao.

Về *Giá thành* DSP được coi là tốt, không rẻ như ASIC nhưng không quá cao so với MCU.

Năng lượng tiêu thụ của DSP được coi là rất tốt, nhất là với loại DSP được thiết kế đặc biệt cho tiêu chí này cho các ứng dụng xách tay như TMS320C5000.

Tính *Dễ phát triển* được đánh giá là rất tốt. Các nhà cung cấp DSP có một mạng lưới thành phần thứ ba cho mọi lĩnh vực để giúp phát triển

DSP, từ các chuyên gia cố vấn cho phần cứng, phần mềm, đến hệ thống.

Cũng vậy, có các công cụ phát triển DSP rất mạnh, dễ sử dụng. Có mạng lưới hỗ trợ kỹ thuật và các kỹ sư am hiểu luôn sẵn sàng giúp đỡ khách hàng đạt được các thiết kế thời gian thực của mình.

Về giá thành phát triển, khả năng lập trình của DSP cho phép chu kỳ phát triển nhanh hơn so với các chip định hướng cho ứng dụng xác định hoặc ASIC. Sử dụng đúng ngôn ngữ lập trình bậc cao kết hợp các module code chuẩn sẽ rút ngắn đáng kể thời gian phát triển, và như vậy tiết kiệm giá thành.

Tính mềm dẻo của DSP là rất tốt, nhất là so với các thực hiện tương tự bằng phần cứng. Đối với xử lý tín hiệu thời gian thực, có nhiều công cụ tốt và thích đáng nhất cũng như có nhiều trang web có giá trị dành cho DSP hơn so với RISC và MCU.

2.1.5 CÔNG NGHỆ FPGA

FPGA (Field Programmable Gate Array): mảng cổng có thể lập trình theo yêu cầu.

Thời gian ra thị trường của FPGA được đánh giá là tốt. Có thể modify các trường của FPGA để được các chức năng khác nhau, nhưng không mềm dẻo như lập trình bằng phần mềm của DSP, MCU và RISC trong góc độ đưa ra thị trường. Tuy nhiên FPGA được hỗ trợ tốt hơn và chu kỳ thời gian nhanh hơn ASSP, các bộ xử lý có thể cấu hình và ASIC và như thế có thể coi tiêu chí *Thời gian ra thị trường* của FPGA tốt hơn.

Năng lực thực hiện của FPGA được đánh giá là rất tốt vì các nhà phát triển có thể vi chỉnh đến các cổng hardware của FPGA cho sát với ứng dụng.

Về *Giá thành* thì FPGA bị coi là kém, đắt hơn nhiều so với 6 loại còn lại.

Về *Năng lượng tiêu thụ* cũng bị đánh giá là kém nhất so với các loại khác do đặc điểm của công nghệ FPGA và do các cổng không dùng đến tiêu thụ năng lượng quá mức. Công nghệ mới ngày nay đã giảm năng lượng tiêu thụ của FPGA nhưng dường như vẫn chưa đủ để xếp FPGA vào hàng ngũ các loại hiệu quả về *Năng lượng tiêu thụ*.

Về *Dễ phát triển,* FPGA được coi là rất tốt. Giá phát triển sẽ là tốt nhất với giả thiết 2 điều kiện: 1) *công cụ lập trình FPGA không quá đắt*; và 2) *các nhà phát triển căn bản phải thông thạo phần cứng*. Nếu các nhà phát triển là các kỹ sư thiên về phần mềm thì phải nỗ lực nhiều và tăng giá thành.

Về hỗ trợ cho phát triển thì các công cụ và cấu trúc cho thiết kế FPGA khá tốt và có khả năng chấp nhận OEM.

Tính mềm dẻo của FPGA được coi là tốt. Nó có thể được cấu hình lại để tăng thêm hoặc thay đổi đặc điểm. Tuy nhiên lập trình lại phần cứng khó hơn và các chức năng thêm cũng hạn chế hơn so với các giải pháp lập trình phần mềm như DSP.

2.1.6 CÔNG NGHỆ MCU

MCU (Microcontroller): vi điều khiển.

Thời gian ra thị trường của MCU được coi là rất tốt, cũng như DSP, RISC. Về xử lý thời gian thực thì mặc dù MCU không thật tốt nhưng do nó được phổ biến rất rộng rãi, cũng như có rất nhiều công cụ và các trang web có giá trị nên MCU được xếp đứng hàng thứ hai.

Về *Năng lực thực hiện* MCU được coi là khá. So với RISC/GPP thì

tài nguyên để thực hiện các phép toán của MCU nhỏ hơn và tần số làm việc cũng chậm hơn.

Giá thành của MCU là rất tốt do nói chung MCU là các chip nhỏ tương đối rẻ và đứng hàng thứ hai sau ASIC.

Về *Năng lượng tiêu thụ* thì MCU được đánh giá là khá. MCU tiêu thụ ít năng lượng hơn RISC và FPGA do nó sử dụng ít tài nguyên silicon hơn. Tuy nhiên kém DSP, ASSP và bộ xử lý có thể cấu hình.

Tính *Dễ phát triển* của MCU được đánh giá là tốt. Khả năng lập trình được của các chip MCU đang có cho phép phát triển các chức năng theo nhu cầu nhanh hơn đối với ASIC hoặc các chip chuyên dụng cho ứng dụng xác định. Sử dụng đúng ngôn ngữ lập trình bậc cao và/hoặc sử dụng các module code chuẩn có thể giảm đáng kể, dẫn đến hạ giá thành phát triển.

Về việc hỗ trợ phát triển, hầu hết các nhà cung cấp MCU đều có mạng lưới giúp đỡ tốt, tuy vậy không được đánh giá là rất tốt vì nhiều mạng này đơn thuần là các ứng dụng nhúng mà không phải ứng dụng thời gian thực.

Tính mềm dẻo MCU được đánh giá là rất tốt, tương tự như các chip có thể lập trình.

2.1.7 CÔNG NGHỆ RISC/GPP

RISC/GPP (Reduced Instruction Set Computer/ General Purpose Processor) - Chip tính toán có tập lệnh rút gọn/Bộ xử lý mục đích chung.

Thời gian ra thị trường của RISC/GPP được coi là tốt, cũng như DSP, các bộ xử lý cấu hình được và MCU. Đối với xử lý tín hiệu thời gian thực, RISC/GPP đứng hàng thứ ba sau MCU. Tuy nhiên nó không đủ

mạnh cho các ứng dụng thời gian thực và không định hướng cho các ứng dụng nhúng mà mục tiêu tập trung vào máy tính để bàn.

Năng lực thực hiện của RISC được coi là tốt. Tần số làm việc cao tăng hiệu suất xử lý tín hiệu. Tuy vậy do không có các lệnh thực hiện phép toán trong một chu kỳ và không có các khối nhân làm việc thực hiện thời gian thực của nó bị giới hạn.

Giá thành của RISC bị coi là khá. Các bộ xử lý RISC tốt với các ứng dụng để bàn, nhưng về mặt giá thì chỉ được coi là khá với xử lý tín hiệu thời gian thực.

Năng lượng tiêu thụ của RISC được đánh giá là khá, xếp dưới so với DSP, ASSP hay các bộ xử lý cấu hình lại.

Dễ phát triển của RISC được đánh giá là tốt. Khả năng lập trình được của RISC đang có cho phép phát triển các chức năng theo nhu cầu nhanh hơn đối với ASIC hoặc các chip chuyên dụng cho ứng dụng xác định. Sử dụng đúng ngôn ngữ lập trình bậc cao và/hoặc sử dụng các module code chuẩn có thể giảm đáng kể, dẫn đến hạ giá thành phát triển.

Về giúp đỡ phát triển, có những hỗ trợ chung cho các nhà thiết kế RISC, nhưng không có những hỗ trợ cho một ứng dụng xác định nào cũng như không có hỗ trợ về thời gian thực do kiến thức của những nhà cung cấp RISC về các nội dung này không mạnh.

Tính mềm dẻo của RISCđược đánh giá là rất tốt so với thực hiện bằng phần cứng, giống như các loại có khả năng lập trình khác.

Có thể tóm tắt tất cả các phân tích trên về 7 công nghệ phổ biến nhất trong xử lý tín hiệu thời gian thực bằng bảng sau.

Bảng 1. So sánh 7 công nghệ phổ biến nhất

	Thời gian ra thị trường	Năng lực thực hiện	Giá thành	Dễ phát triển	Năng lượng tiêu thụ	Tính mềm dẻo	Đánh giá chung
ASIC	Kém	Rất tốt	Rất tốt	Khá	Tốt	Kém	Khá
ASSP	Khá	Rất tốt	Tốt	Khá	Rất tốt	Kém	Tốt
VXL có thể cấu hình	Kém	Rất tốt	Tốt	Kém	Tốt	Khá	Khá
DSP	Rất tốt	Rất tốt	Tốt	Rất tốt	Rất tốt	Rất tốt	Rất tốt
FPGA	Tốt	Rất tốt	Kém	Rất tốt	Kém	Tốt	Khá
MCU	Rất tốt	Khá	Rất tốt	Tốt	Khá	Rất tốt	Tốt
RISC/GPP	Tốt	Tốt	Khá	Tốt	Khá	Rất tốt	Tốt

Qua bảng trên có thể thấy, tùy thuộc vào tiêu chí nào được nhấn mạnh mà ta có thể chọn công nghệ này hay công nghệ khác. Tuy nhiên tổng thể chung thì DSP là lựa chọn tốt nhất *cho ứng dụng thời gian thực*.

2.1.8 SỬ DỤNG DSP TRONG MẬT MÃ

Những phân tích phần trên đã dẫn đến kết luận "*DSP là lựa chọn tốt nhất cho ứng dụng thời gian thực*". Tuy nhiên đối với mật mã, ngoài yêu cầu mã/giải mã trong thời gian thực, còn yêu cầu thứ hai cũng không kém phần quan trọng, đó là an toàn mật mã[4] (security), hay chúng ta vẫn quen gọi là an toàn nghiệp vụ mật mã. Ta hãy xem xét về tính an toàn của DSP [4].

DSP có ưu thế vượt trội các công nghệ khác ở điểm thực hiện các phép nhân nhanh hơn, nhất là phép nhân các số nguyên lớn. Giới hạn chính của DSP là vấn đề an toàn của nó. DSP hoạt động kiểu mở khi nhận tín hiệu từ đầu vào và xuất trên đầu ra qua bus mà bus lại có thể truy nhập công khai. Cơ chế này làm tăng cao độ rủi ro cho các hoạt động trên khóa riêng. Trong mật mã khóa công khai sử dụng một số phép nhân, những phép toán này tạo ra các giá trị tạm thời trên bus. Do vậy có thể dễ dàng

[4] Lưu ý là tại thời điểm bài này được viết (năm 2004), chúng ta vẫn còn chưa chú trọng đến vấn đề này.

sử dụng các giá trị này để thám ra khóa riêng. Thêm nữa đối phương có thể sửa đổi các giá trị đó trước khi đưa vào DSP để giả mạo chữ ký.

Như thế, xét ở góc độ an toàn thì DSP không thích hợp cho mật mã. Nhưng trong một thiết bị mật mã thì không phải mọi module đều liên quan đến tính toán khóa. Chúng ta có thể sử dụng thế mạnh của DSP chỉ trong các module thuần túy xử lý tín hiệu thời gian thực, như [4] đã khuyến cáo.

Những phân tích trên cũng đúng với các bộ xử lý mục đích chung RISC/GPP và mức độ nào đấy với cả MCU.

Vậy nếu DSP, RISC/GPP, MCU và các Bộ xử lý có thể cấu hình không thích hợp để thiết kế crypto module, thì công nghệ nào sẽ được chọn? Câu hỏi này sẽ được giải đáp trong phần tiếp theo.

2.2 CÔNG NGHỆ FPGA

Tất nhiên vẫn có thể sử dụng DSP cũng như các chip có lập trình nói chung cho thiết kế các module mật mã nhưng phải áp dụng các thiết kế đặc biệt đảm bảo an toàn chống xâm nhập trái phép... Tuy nhiên các giải pháp đó thuộc về phạm vi nghiên cứu khác. Trong tài liệu này chúng tôi muốn đề cập đến các công nghệ tự nó đã mang những thuộc tính thích hợp nhất cho mật mã mà không phải các thiết kế đặc biệt hỗ trợ thêm.

Như Phần 2.1 đã cho thấy, mặc dù DSP, RISC/GPP, MCU và các Bộ xử lý có thể cấu hình rất mạnh về xử lý tín hiệu thời gian thực, nhưng dùng để thiết kế crypto module thì không thích hợp ở khía cạnh an toàn mật mã. Còn ASIC, ASSP và FPGA tự nó đã mang những thuộc tính thích hợp cho an toàn mật mã (như cấu trúc chip được đốt vật lý, bảo đảm toàn vẹn, chống tấn công thám thiết kế, không phụ thuộc vào hệ điều hành nào...). Bởi vậy chúng ta sẽ giới hạn xem xét ở 3 công nghệ này. Có các

ngữ cảnh cụ thể như sau:

- Đối với những sản phẩm đang trong quá trình nghiên cứu phát triển: tiêu chí *Dễ phát triển* góp phần đáng kể rút ngắn thời gian nghiên cứu phát triển cũng như hạ giá thành tổng thể cho các sản phẩm loạt nhỏ và vừa → FPGA là thích hợp.
- Đối với những sản phẩm cần khả năng cấu hình lại (như thay thế thuật toán) thì *Tính mềm dẻo* cần chú trọng → FPGA là thích hợp.
- Đối với những sản phẩm cần tiêu thụ năng lượng thấp (như trong các thiết bị mang xách) và không đòi hỏi tính mềm dẻo → ASSP là thích hợp.
- Đối với những sản phẩm đã xong về thiết kế, không đòi hỏi tính mềm dẻo và sản xuất loạt lớn → ASIC là thích hợp.

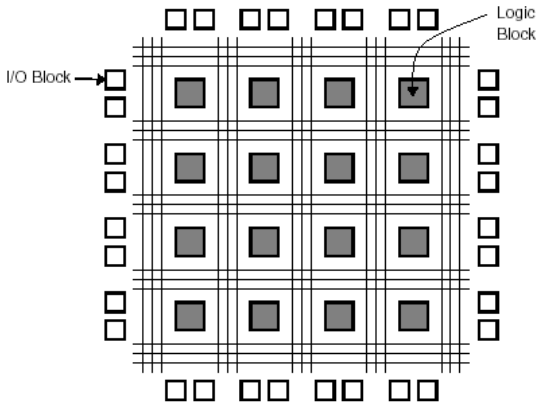
Đối với ngành mật mã thì *Tính mềm dẻo* cũng là một tiêu chí phải xếp lên hàng đầu bởi thuật toán sinh khóa/mã hóa có thể thay đổi theo từng phiên liên lạc hoặc khi chuyển mạng. ***Bởi vậy công nghệ thích hợp nhất để cứng hóa mật mã để lựa chọn chính là FPGA*** với những ưu điểm chính là:

- Tốc độ cao vì nó thực sự là hardware
- An toàn mật mã cao
- Mềm dẻo
- Dễ phát triển

2.2.1 CẤU TRÚC FPGA

FPGA gồm dãy *các phần tử mạch* độc lập gọi là các khối logic, và

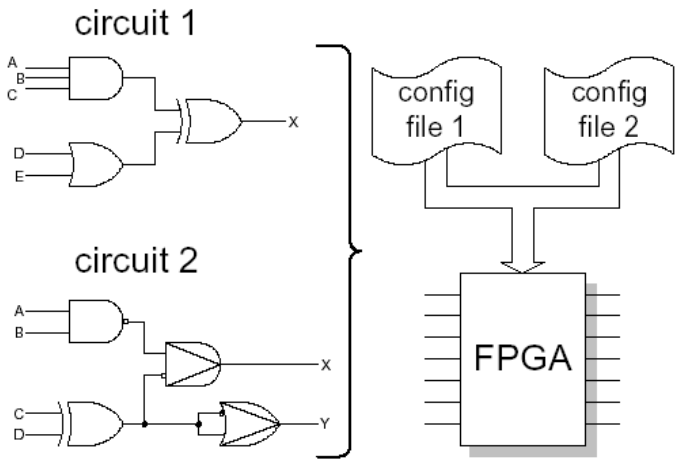
các đường nối các khối đó với nhau [3], [7]. Các khối logic và các đường nối giữa chúng được gọi là tài nguyên. Mỗi khối logic chứa Boolean logic và các thanh ghi. Có thể lập trình cho mỗi khối logic để được các chức năng khác nhau. Các đường nối giữa các khối cũng được lập trình. Thiết lập cấu hình cho FPGA, hiểu một cách đại thể, là lập trình nối các khối logic theo cách nào đó để được một cấu trúc mạch thực hiện thuật toán đã cho. Công việc này do người dùng cuối làm bằng cách lập trình. Hình 2 minh họa cấu trúc của một FPGA điển hình.



Hình 2. Cấu trúc FPGA

2.2.2 KHẢ NĂNG CẤU HÌNH LẠI CỦA FPGA

FPGA thuộc về các chip có thể lập trình theo yêu cầu [7]. Chúng gồm ba loại chính: đơn giản (SPLD), phức tạp (CPLD) và chuỗi các cổng có thể lập trình theo nhu cầu Field-Programmable Gate Arrays (FPGAs), trong đó FPGA là loại mạnh nhất, nhiều tài nguyên, có thể thực hiện các bài toán phức tạp trong mật mã và có *Khả năng cấu hình lại*. Một chip FPGA có thể đóng các vai trò khác nhau tùy theo file cấu hình nào được nạp vào cho nó [11]. Hình 3 minh họa khả năng cấu hình lại của FPGA.



Hình 3. Minh họa khả năng cấu hình lại của FPGA

2.2.3 NHỮNG ƯU ĐIỂM CỦA FPGA ĐỐI VỚI MẬT MÃ

Những đặc tính của FPGA cùng khả năng cấu hình lại làm PPGA trở nên thuận lợi lớn khi thực hiện các thuật toán mật mã [3], [19]. Đó là:

1. *Dễ dàng chuyển thuật toán:* Các giao thức mật mã hiện đại là các thuật toán độc lập, rất đa dạng, thay đổi theo mỗi phiên (chẳng hạn như DES, 3DES, Blowfish, CAST, IDEA, RC4, RC6, AES...). Với khả năng cấu hình lại, FPGA cho phép các thuật toán, sau khi đã cứng hóa, vẫn có khả năng thay đổi *ngay trong khi đang chạy* mà không làm tăng giá thành.

2. *Dễ dàng upgrade thuật toán:* upgrade thuật toán cần thiết trong các trường hợp sau: khi thuật toán hiện tại đã bị phá (ví dụ DES); khi thuật toán đã quá hạn (ví dụ DES); khi ra đời thuật toán mới (ví dụ AES); khi danh sách các giao thức độc lập về thuật toán được mở rộng; khi đối tác liên lạc xa như thông tin vệ tinh.

3. *Mang lại hiệu quả về cấu trúc:* trong trường hợp nhất định, một cấu trúc hardware có thể hiệu quả hơn nhiều nếu nó được thiết kế với một bộ các thông số xác định. Chẳng hạn phép nhân hằng (của

các số nguyên trong trường Galois) hiệu quả hơn các phép nhân thông thường nhiều. Với FPGA có thể thiết kế để tối ưu cấu trúc cho bộ tham số xác định của từng thuật toán khác nhau

Ví dụ 1: Cấu trúc khóa xác định: với khóa cố định thì thao tác chính trong IDEA suy biến thành các phép nhân hằng hiệu quả hơn phép nhân thường nhiều.

Ví dụ 2: Phép toán số học trên trường Galois cố định: Phép toán số học trên trường Galois hiệu quả hơn nhiều nếu bậc của trường và đa thức bất khả quy được cố định. Việc này FPGA giải quyết tốt.

Ví dụ: Phép bình phương trong GF(2^m) chiếm $m/2$ chu kỳ với cấu trúc thông thường. Nhưng với cấu trúc một trường cố định thì chỉ cần 1 chu kỳ.

4. *Hiệu quả về tài nguyên:* Có thể cấu hình lại FPGA để *cùng một chip* nhưng mỗi thời điểm thực hiện một thuật toán khác nhau. Ví dụ trong một phiên liên lạc, khi thiết lập khóa chung FPGA có cấu trúc để thực hiện thuật toán khóa công khai; khi thực hiện mã dịch, FPGA có cấu trúc để thực hiện mã dịch sử dụng thuật toán khóa riêng.

5. *Khả năng modify thuật toán:* với ngành mật mã chúng ta thì yêu cầu modify thuật toán là bắt buộc, ví dụ thay các module S-box hay các hoán vị chuẩn bằng các S-boxes hay hoán vị độc quyền; một số ứng dụng khác lại cần thay đổi chế độ làm việc (chế độ hối tiếp, chế độ đếm...); còn các máy tìm khóa trong mã thám cho phép sử dụng phiên bản khác chút ít với thuật toán. Với FPGA, tất cả những điều đó đều dễ dàng thực hiện.

Hai đặc điểm dưới đây không chỉ với mật mã mà chung cho mọi

lĩnh vực khác:

6. *Thông lượng:* mặc dù so với ASIC thì FPGA chậm hơn, nhưng nhanh hơn rất nhiều so với software.

7. *Hiệu quả về giá thành:* thời gian và giá thành phát triển của FPGA thấp hơn so với ASIC. (Tuy nhiên với số lượng lớn thì ASIC có giá thành thấp hơn).

2.3 THỰC HIỆN MẬT MÃ BẰNG FPGA

2.3.1 THỰC HIỆN MẬT MÃ ĐỐI XỨNG BẰNG FPGA

Mật mã khóa đối xứng phụ thuộc vào khóa riêng. Phép mã hóa có thể chỉ bằng phép XOR đơn giản. Trong trường hợp khóa trong thì khóa riêng sẽ do thuật toán sinh ra. Nhiều thuật toán sinh khóa có cấu trúc gồm các thanh ghi dịch phản hồi phối hợp với nhau theo hàm phi tuyến nào đó. Hoạt động của các thanh ghi dịch, nếu bằng giải pháp software, thường chiếm khá nhiều chu kỳ lệnh do giới hạn số bit của từ xử lý của các CPU. Với giải pháp hardware nói chung và FPGA nói riêng, có thể thiết kế thanh ghi dịch có độ dài theo yêu cầu, chỉ với một chu kỳ đồng hồ là cho ra một bít. [8] đã đưa ra kết quả thực hiện một thuật toán mã dòng bằng FPGA đạt tốc độ 4.6 Gbps.

2.3.2 THỰC HIỆN MẬT MÃ KHÔNG ĐỐI XỨNG BẰNG FPGA

Tâm điểm của mật mã không đối xứng là các phép toán số học mũ hóa modular với số nguyên lớn, điển hình là thuật toán trao đổi khóa Diffie-Hellman, mã/giải mã RSA, chữ ký số DSA. Vấn đề đặt ra là làm sao thiết kế được các cấu trúc số học với các toán tử lên đến 1024 bit và hơn nữa trên FPGA. Theo cách thiết kế thông thường thì tốc độ thực hiện cũng như tài nguyên của các chip FPGA hiện tại là không đủ. Bởi vậy đã có nhiều nghiên cứu nhằm tăng tốc độ và tăng hiệu quả sử dụng tài nguyên FPGA khi thực hiện các thuật toán mật mã công khai [8], [9],

[10], [11], [12], [13], [14], [15], [16]...

Chúng ta sẽ lấy trao đổi khóa Diffie-Hellman làm ví dụ để phân tích về thực hiện mật mã không đối xứng bằng FPGA. Thuật toán đang được quan tâm và được coi là rất thích hợp đối với FPGA khi thực hiện các phép mũ modular là Montgomery bởi nó cho giá thành thấp và không cần bộ nhân đặc biệt.

Thuật toán Montgomery cho phép thực hiện phép nhân modular mà không phải thực hiện phép chia. Đây là điều quan trọng vì thực hiện phép chia bằng phần cứng rất khó. Tương tự như với số nguyên thông thường, có thể dùng thuật toán bình phương và nhân để thực hiện phép mũ các số Montgomery. Khi sử dụng các số Montgomery, chỉ cần thêm hai bước: đầu tiên chuyển các số nguyên sang không gian Montgomery với tất cả các phép nhân trung gian được thực hiện khi sử dụng kỹ thuật Montgomery. Sau đó chuyển kết quả trở lại dạng số nguyên thông thường.

Phép nhân Montgomery – *AB mod M* – cho cơ số 2 (tức là tính một số *n*-bit cần *n* phép lặp) là:

1. $S_{-1} = 0$
2. FOR i = 0 to n-1 loop
3. $q_i = (S_{i-1} + b_i A) \bmod 2$
4. $S_i = \dfrac{(S_{i-1} + q_i M + b_i A)}{2}$
5. END FOR
6. RETURN S_{n-1}

ở đây chỉ số dưới *i* là vị trí bit riêng rẽ trong một số, với *i* = 0 là bit có nghĩa thấp nhất (LSB).

Trong phép nhân Montgomery, đặc biệt là phép dịch phải, có một thừa số trong kết quả. Vậy kết quả sẽ là:

$$S = A \cdot B \cdot 2^{-n} \bmod M$$

Do đó cần phải ánh xạ các toán tử *A* và *B* vào không gian Montgomery, gọi là *m*- thặng dư, bằng phép modular:

$$A' = A \cdot 2^n \bmod M$$

Do cả hai toán tử đều được chuyển thành *m*- thặng dư nên có thừa số 2^n trong kết quả Montgomery cuối cùng. Thừa số này sẽ được gỡ bỏ với phép nhân Montgomery cuối cùng bằng số nguyên '1'.

Khi thực hiện trên FPGA, hạt nhân của phép nhân modular Montgomery là phép tính trung gian:

$$S_i = \frac{(S_{i-1} + q_i M + b_i A)}{2}$$

Ba thừa số: số trung gian *S*, modulus *M*, và toán tử *A* đều có độ lớn *n* bit. Ngày nay yêu cầu *n* phải đạt 1024 hay 2048. Do công nghệ FPGA hiện nay không thể cộng một cách có hiệu quả các số có kích thước lớn như vậy nên cần một phương pháp gián tiếp.

Một trong các phương pháp đó là sử dụng cả dãy systolic và bộ cộng pipeline. Tuy nhiên nếu chia phương trình trên thành các phần nhỏ và tính toán lặp lại, thì có thể sử dụng tài nguyên của FPGA hiệu quả hơn, đặc biệt là tài nguyên đường nối.

Thường thì người ta hay đánh giá khả năng thực hiện thuật toán FPGA bằng tài nguyên logic. Tuy nhiên tài nguyên dây nối cũng liên quan chặt chẽ với kết quả cuối theo hai cách: thứ nhất là khả năng thực hiện của chính hạt nhân và thứ hai là logic bao quanh hệ thống mà hạt nhân là một phần của nó. Nếu tỷ lệ sử dụng đường dẫn lớn hơn sử dụng logic thì phải bổ sung thêm tài nguyên vào hệ thống dẫn đến giảm hiệu suất thực hiện toàn hệ thống.

Bằng cách tính các giá trị trung gian theo kiểu liên tiếp, nhân logic và đường dẫn có thể được chứa trong phần rất nhỏ, dễ nối của chip, dẫn

đến nhân có thể làm việc tại tốc độ cao nhất mà chip cho phép.

Các tham số *S, M, B* và *A* được lưu trong bộ nhớ, để cung cấp giá trị cho từng phép nhân Montgomery. Ngoài ra còn phải lưu số mũ *E* và giá trị mũ modular trung gian. Một máy trạng thái điều khiển các phép nhân đệ quy để tính số mũ modular.

Việc thâm nhập bộ nhớ trực tiếp thông qua giao diện có độ rộng của một từ. Người sử dụng có thể đặt tham số để bus hệ thống (rộng16, 32, 64 hay 128 bit tùy theo hệ thống) tương xứng với độ rộng của giao diện. Thông lượng của nhân tỉ lệ trực tiếp với độ rộng từ do người dùng xác lập.

Phần trên đã xem xét về cấu trúc của nhân sao cho thực hiện giao thức Diffie-Halman bằng FPGA một cách tốt nhất với hiệu suất thực hiện cao nhất và tài nguyên ít nhất có thể. Ngoài các phép toán ra, để thực hiện giao thức còn cần một số hoạt động sau:

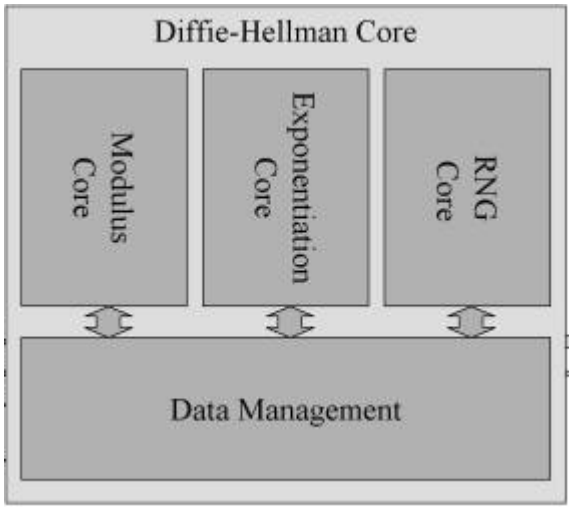
- Chọn số ngẫu nhiên thống kê cần cho giai đoạn đầu của giao thức.
- Tạo các gói để hệ thống sử dụng
- Kiểm tra các số ngẫu nhiên và khóa để tránh tấn công va chạm khi gửi đi các khóa lặp lại hay khóa yếu.
- Tạo ra các gói với phương pháp xác thực nào đó để giảm nguy cơ tấn công xen giữa.

Trong thực tế giao thức Diffie-Hellman là tổ hợp của ít nhất ba thành phần cơ bản:

1. Khối số học modular
2. Bộ sinh số ngẫu nhiên (RNG).

3. Bộ xử lý điều khiển luồng số liệu và xử lý các gói. Có thể xử dụng luôn bộ xử lý nhúng trong FPGA hay bộ xử lý ngoài.

Hình 4 mô tả một module tăng tốc Diffie-Hellman điển hình có thể sử dụng trong FPGA. Trong hình chỉ có một khối tính toán, tuy nhiên hoàn toàn có thể tăng số lượng khối này lên để tăng thông lượng hệ thống lên gấp bội một cách tuyến tính.



Hình 4. Khối tăng tốc Difie-Hellman bằng FPGA

Khi thực hiện một hàm RSA bằng phần cứng, cần có một hàm modulus đơn giản để chuyển đổi không gian *m*- thặng dư trước khi bắt đầu phép mũ Montgomery, hàm đó như sau:

$Y = A \bmod M$

Cũng cần có hạt nhân modulus để tăng tốc tính toán các giá trị trung gian khi thực hiện mũ hóa modular dựa trên định lý phần dư China: phân một phép mũ modular thành các phép mũ modular nhỏ. Do độ phức tạp của phép mũ modular bằng bình phương độ dài từ nên việc này sẽ cho hiệu suất thực hiện gấp bốn lần cách tính thông thường.

Trong hạt nhân Difie-Hellman, hàm modulus này có thể nhằm phục

vụ mục đích khác ngoài phép chuyển đổi *m*-thặng dư. Khi số nguyên thủy $g = 2$, thì việc tính toán đầu tiên của Diffie-Hellman là $Y = s^x \bmod p$. Để biểu diễn lũy thừa của 2 dưới dạng binary, cần một phép ước lượng đơn giản bằng cách chèn '1' vào vị trí thích hợp trong trường các số zero, kết quả là phép toán trở thành $Y = A \bmod p$ trong đó $A = 2^x$, điều này tương đương với biểu thức miêu tả hạt nhân modulus về mặt hàm số.

Phép đơn giản hóa này là quan trọng trong giai đoạn đầu sinh khóa vì nó suy biến toàn bộ phép mũ modular thành một modulus duy nhất. Do phép modulus, trong một số mức độ, nhanh hơn phép mũ tương đương nên biện pháp đơn giản này giúp tiết kiệm được đáng kể thời gian tính toán.

Hạt nhân dùng để tính toán Diffie-Helman là một phần của toàn hệ thống. Sau khi đã trao đổi xong khóa, có thể sử dụng nó cho việc khác hoặc sẵn sàng cho lần trao đổi khóa tiếp theo.

So với giải pháp tính toán trao đổi khóa bằng phần mềm dựa trên vi xử lý mục đích chung GPP hay DSP, giải pháp tăng tốc bằng phần cứng dựa trên FPGA kém hơn về tính mềm dẻo, nhưng ưu điểm hơn ở các mặt sau:

1. Số lượng khóa tính được trong một giây nhiều hơn hẳn.
2. Không cần một bộ xử lý chuyên trách cho tác vụ trao đổi khóa, dẫn đến giảm kích thước bảng mạch, công suất tiêu thụ, thời gian thiết kế $\rightarrow$ giảm giá thành hệ thống.

Một kết quả của việc sử dụng FPGA kết hợp bộ xử lý nhúng trong chip FPGA đã được [15] dẫn như sau: với số mũ và modulus 1024 bit, bộ mũ hóa cơ số 2, sử dụng 500 cell logic thì tính được 32 khóa trong 1 giây. Còn với FPGA mạnh hơn có bộ nhân 36 x 36, hỗ trợ phép mũ hóa

modular cơ số cao hơn, chỉ với lượng nhỏ tài nguyên của chip, có thể đạt đến 640 khóa trong một giây. Hiệu suất thực hiện và giá thành có thể so sánh với ASSP, nhưng lại có tính mềm dẻo của bộ xử lý; và quan trọng hơn là, giải pháp FPGA cho phép người dùng có thể tùy biến theo yêu cầu để tối ưu sản phẩm cuối cùng.

Một lợi ích nữa của giải pháp FPGA là có thể thiết kế giao thức Diffie-Haellman và các số nguyên tố phi chuẩn độc quyền. Ví dụ thiết kế một nhóm Diffie-Hellman có phần tử nguyên thủy $g = 13$. Có thể thay đổi phần tử nguyên thủy này dễ dàng như thay đổi các tham số do người dùng định nghĩa trong file cấu hình của hạt nhân tăng tốc hay thậm chí update cả platform phần cứng.

2.3.3 THỰC HIỆN AES BẰNG FPGA

Các thuật toán AES tiêu thụ tương đối nhiều tài nguyên phần cứng khi thiết kế trên FPGA do tính phức tạp của chúng. Kết quả cuối cùng (về hiệu suất sử dụng tài nguyên, tốc độ) sẽ rất khác nhau khi sử dụng các phương pháp thiết kế khác nhau (dựa trên ngôn ngữ mô tả phần cứng hay dựa trên sơ đồ) và các công cụ thiết kế (các Kit phát triển, các phần mềm tổng hợp và mô phỏng của các hãng khác nhau). Điều này cũng tương tự như lập trình cho máy tính bằng ngôn ngữ bậc cao nào đó: các trình compiler của các hãng khác nhau sẽ kết xuất các file thực hiện có kích thước khác nhau và tốc độ thực hiện cũng khác nhau.

2.3.3.1 YÊU CẦU CHIP FPGA ĐỂ THỰC HIỆN AES

Tài nguyên chip FPGA là vấn đề đầu tiên cần xem xét khi thiết kế cho AES. Có thể sử dụng nhiều chip FPGA cho thiết kế một thuật toán, tuy nhiên tăng giá thành. Vì thế người ta thường cố đến thực hiện toàn bộ thuật toán AES trong *một* chip FPGA. Khi ấy phải lưu tâm đến các vấn đề tài nguyên của chip [19]. Trước tiên, để đạt được tốc độ cao cần số lượng

lớn các chân vào ra I/O nhằm hỗ trợ hoàn toàn luồng data 128-bit. Cũng không nên dùng giải pháp dồn kênh mà phải tách 128 bit data input và output ra khỏi nhau để cho phép cấu trúc pipeline.

Tiếp theo, phải có bus riêng để nhập khóa trước khi bắt đầu mã hóa vì không thể thay đổi khóa của các vòng trong khi đang mã. Thêm nữa để thực hiện cấu trúc pipeline một cách hoàn toàn cần các tầng pipeline rộng 128 bit, như thế lại cần rất nhiều thanh ghi. Và nữa, cũng cần nhiều bit ghi trong các khối có thể cấu hình của chip FPGA để sắp xếp các phần tử thiết kế cũng như giảm thiểu số đường dẫn giữa các khối đó. Cuối cùng, cần chuỗi carry nhanh giữa các khối có thể cấu hình của FPGA cho các thao tác số học để cực đại hiệu suất thực hiện các thuật toán AES.

Chip FPGA có thể đáp ứng được các yêu cầu trên có thể là Xilinx Virtex XCV1000BG560-4. XCV1000 có 128K bits RAM nhúng lấy từ 3/2 block RAM của chip FPGA. Chip được đóng trong vỏ 560 chân, cung cấp 512 chân vào ra I/O; có một mảng gồm 64 x 96 khối logic có thể cấu hình CLB, tạo nên bảng tìm kiếm. Mỗi khối CLB có hai mảnh 2 bit và hoạt động như một phần tử 4 bit . Tổng cộng có 12288 mảnh CLB. Ngoài RAM nhúng, XCV1000 có thể được cấu hình để có 384 K bits RAM khác nữa. Kiểu cấu hình này cho phép cấu trúc có độ phức tạp cao phù hợp với chức năng vòng lặp của các hàm có toán hạng lớn.

2.3.3.2 CẤU TRÚC HARDWARE FPGA ĐỂ THỰC HIỆN AES

Các AES đều có cấu trúc vòng, số liệu quay vòng liên tiếp qua một hàm lặp. Để tối ưu thực hiện cấu trúc vòng có thể có các cách sau [19]:

- Lặp liên tiếp
- Lặp Unrolling
- Pipeline một phần

- Pipeline một phần với Sub-Pipeling

Bộ mã hóa có cấu trúc lặp liên tiếp là phương pháp hiệu quả tiêu thụ ít tài nguyên hardware nhất. Bộ mã hóa *n* vòng cần lặp liên tiếp *n* lần cho một thao tác mã hóa. Giải pháp này có thời gian trễ từ thanh ghi - đến-thanh ghi nhỏ nhưng phải mất nhiều chu kỳ đồng hồ cho một thao tác mã. Thêm nữa, tuy tiêu thụ ít tài nguyên phần cứng nhưng có thể giá thành vẫn tăng do yêu cầu phần cứng khác để thực hiện khóa của vòng và S-Box multiplexing.

Có thể giảm được giá thành này bằng cách tạo một bảng tìm kiếm 4 bit trong mảnh CLB để làm việc trong mode RAM và cất giữ một bit của mỗi khóa vòng trong bảng đó. Một bảng tìm kiếm hỗ trợ lên đến 16 khóa vòng dẫn đến việc thực hiện *k* bảng tìm kiếm để lưu trữ các khóa vòng có độ dài *k* bit, tránh phải lưu các khóa vòng trong một thanh ghi riêng. Thêm nữa, nhiều băng của bảng tìm kiếm có thể được dùng để hỗ trợ cho các thuật toán yêu cầu nhiều hơn 16 khóa vòng. Các băng này sau đó sẽ được đánh địa chỉ liên tục dựa trên vòng hiện tại để truy nhập khóa thích hợp. Tuy nhiên, phương pháp này kém hiệu quả hơn khi nhiều vòng được unroll hay pipeline vì mỗi vòng cần băng riêng của mình trong bảng tìm kiếm để lưu trữ khóa vòng. Đối với cấu trúc unroll hay pipeline toàn bộ phương pháp này hiệu quả hơn vì mỗi băng của bảng tìm kiếm chỉ chứa một khóa vòng duy nhất.

Vòng lặp lập lại là một tập con của vòng lặp unroll trong đó chỉ một vòng được unroll, ngược lại cấu trúc unroll lặp cho phép unroll nhiều vòng với số lượng theo nhu cầu của bộ mã hóa. Trái với cấu trúc lặp lập lại, trong cấu trúc lặp unroll tất cả *n* vòng được unroll và thực hiện như một khối logic tổ hợp duy nhất. Điều này đòi hỏi rất nhiều tài nguyên phần cứng để thực hiện chức năng vòng trong khi phần cứng cần cho khóa

của vòng và S-Box multiplexing chỉ có hạn. Mặc dù giải pháp này cần tối thiểu số chu kỳ clock để thực hiện một mã hóa, nhưng nó lại có thời gian trễ từ thanh ghi đến thanh ghi xấu nhất, làm hệ thống chạy cực kỳ chậm.

Cấu trúc pipeline một phần có ưu điểm cho thông lượng cao nhờ tăng số khối data được xử lý đồng thời. Điều này đạt được bằng cách tạo một bản sao hàm vòng bằng phần cứng và ghi số liệu trung gian giữa các vòng. Thêm nữa, trong trường hợp pipeline toàn bộ độ dài (một dạng đặc biệt của pipeline một phần), mỗi chu kỳ clock hệ thống sẽ xuất ra một khối mã 128 bit mỗi khi pipeline có thể có. Tuy nhiên cấu trúc dạng này cần nhiều tài nguyên phần cứng hơn hẳn cấu trúc unroll lặp. Trong cấu trúc pipeline một phần, mỗi vòng được thực hiện như một đơn vị nguyên tử của pipeline và được tách riêng bởi các thanh ghi của pipeline thực sự. Đối với một bộ mã hóa n vòng, cả các hàm và hộp này đều phải được sao ra *n* lần. Như thế, các thuật toán phải được thực hiện như là các pipeline một phần. Thêm nữa, cấu trúc pipeline chỉ có thể được dùng toàn bộ trong chế độ làm việc không đòi hỏi hồi tiếp dữ liệu mã. Trong chế độ hồi tiếp, dữ liệu của một khối phải được mã xong trước khi mã khối tiếp theo, kết quả là không thể mã nhiều khối rõ theo kiểu pipeline.

Khi hàm vòng của cấu trúc pipeline phức tạp gây ra độ trễ lớn giữa các tầng pipeline, thì có thể phân nhỏ cấu trúc bằng cách thêm các tầng pipeline nhỏ, hàm nguyên tử của mỗi tầng pipeline được chia thành các khối chức năng nhỏ hơn. Kết quả giảm được độ trễ của pipeline giữa các tầng. Tuy nhiên mỗi sự chia nhỏ hàm nguyên tử lại làm tăng số chu kỳ đồng hồ cần thiết để thực hiện một phép mã hóa thêm một hệ số bằng số phép chia. Cùng lúc, số khối data có thể xử lý trong chế độ không hồi tiếp cũng tăng một hệ số bằng số các phép chia. Do đó, để kỹ thuật này hiệu quả, trường hợp xấu nhất độ trễ giữa các tầng sẽ được giảm một hệ số *m* là số các phép chia thêm vào.

Nhiều FPGA có RAM nhúng có thể được dùng để thực hiện khóa vòng và S-Box multiplexing thay cho phần cứng. Bằng cách lưu trữ khóa trong các khối RAM, có thể đánh địa chỉ khóa thích hợp dựa trên vòng hiện tại. Tuy nhiên do số lượng khối RAM chỉ có hạn, cũng như độ rộng bit của chúng giới hạn nên phương pháp này không thể thực hiện được trong các cấu trúc có nhiều tầng pipeline hay nhiều vòng lặp unroll. Các cấu trúc này cần nhiều khối RAM mà một FPGA thông thường không đáp ứng được[5]. Cần cân nhắc giải pháp dùng RAM nhúng vì thời gian chuyển mạch của RAM lâu hơn 3 lần so với thực hiện bằng phần tử mảnh CLB chuẩn.

2.3.3.3 MỘT SỐ ĐÁNH GIÁ AES KHI THIẾT KẾ TRÊN FPGA

Trước khi NIST quyết định chọn ứng viên nào làm AES chính thức, đã có nhiều đánh giá năm AES vào chung kết do nhiều nhóm khác nhau độc lập thực hiện trên FPGA. Các đánh giá tập trung vào thông lượng của mỗi thuật toán - yêu cầu bắt buộc cho các ứng dụng bảo mật kênh băng rộng hiện tại và tương lai, và tài nguyên hardware tiêu tốn - yêu cầu về giá thành. Hai tiêu chuẩn này quan hệ với nhau theo:

TPS = (Tốc độ mã hóa) / (Số các CLB sử dụng)

Trong đó TPS là thông lượng trên một Slice, CLB là khối logic có thể cấu hình.

Dưới đây là kết luận của các nhóm đánh giá AES được tổng hợp và báo cáo trong [18] và [19].

Kết luận của [19]

Trong đó các ký hiệu được giải nghĩa như sau:

[5] Đây là nói tại thời điểm bài báo này được viết. Lưu ý là với sự phát triển rất nhanh của công nghệ linh kiện điện tử thì trong tương lai gần dung lượng RAM nhúng này chắc chắn sẽ đáp ứng được.

Opt: tối ưu theo tốc độ hay theo area.

LU (Loop Unrolling): cấu trúc kiểu vòng lặp unroll.

PP (full or Partial Pipelining): cấu trúc kiểu pipeline toàn bộ hay một phần.

SP (sub-pipelining): cấu trúc kiểu pipeline một phần với sub-pipeline.

Các chỉ số tiếp theo là số tầng và (nếu cần) số các tầng sub-pipeline.

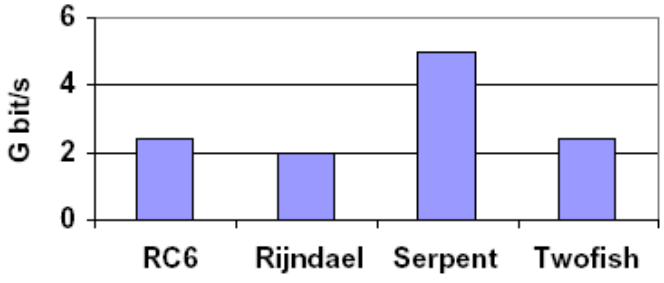
Ví dụ:

LU-4 là cấu trúc vòng lặp unrolling với 4 vòng.

SP-2-1 là cấu trúc pipeline một phần, có 2 tầng và 1 tầng sub-pipeline trên một tầng pipeline. Kết quả là cấu trúc SP-2-1 thực hiện hai vòng mã hóa với tổng cộng 2 tầng trên một vòng.

Bảng 2.1 Thông lượng của các AES trong chế độ không hồi tiếp

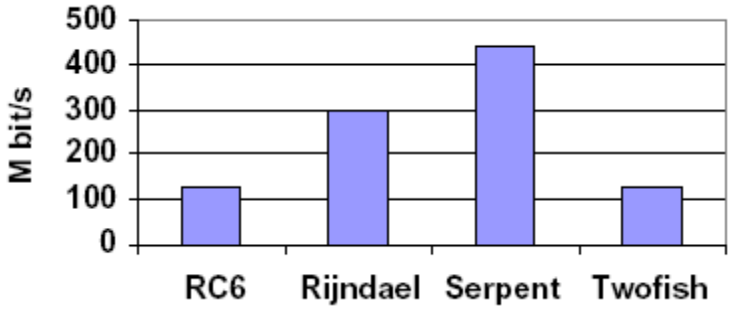
Thuật toán	**Cấu trúc**	**Tối ưu**	**Số chu kỳ trên một khối mã**	Thông lượng (Gbit/s)
RC6	SP-10-2	Tốc độ	2	**2.40**
Rijndael	SP-5-1	Tốc độ	2.1	**1.94**
Serpent	PP-32	Area	1	**5.04**
Twofish	**SP-8-2**	**Tốc độ**	**2**	**2.40**



Hình 2.1 Thông lượng của các AES trong chế độ không hồi tiếp

Bảng 2.2 Thông lượng của các AES trong chế độ hồi tiếp

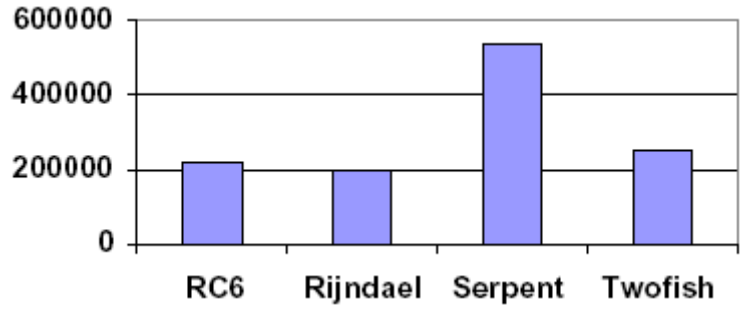
Thuật toán	Cấu trúc	Tối ưu	Số chu kỳ trên một khối mã	Thông lượng (Mbit/s)
RC6	PP-2	Tốc độ	20	**126.5**
Rijndael	LU-2	Tốc độ	6	**300.1**
Serpent	LU-8	Tốc độ	4	**444.2**
Twofish	**SP-1-1**	**Area**	**32**	**127.7**



Hình 2.2 Thông lượng của các AES trong chế độ hồi tiếp

Bảng 2.3 TPS của các AES trong chế độ không hồi tiếp

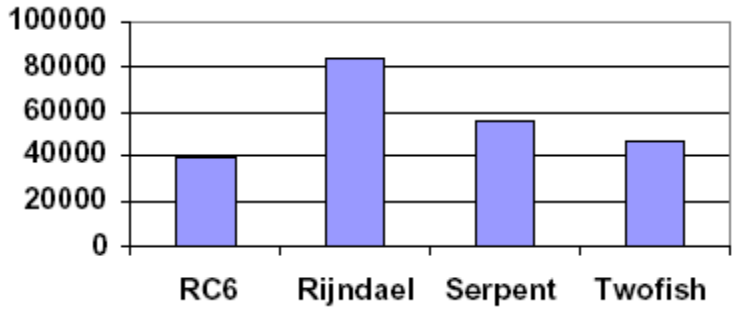
Thuật toán	Cấu trúc	Tối ưu	Slices	TPS
RC6	SP-10-2	Tốc độ	10856	**220881**
Rijndael	SP-2-1	Tốc độ	4871	**194837**
Serpent	PP-32	Tốc độ	2112	**539778**
Twofish	**SP-8-2**	**Area**	**672**	**248666**



Hình 2.3 TPS của các AES trong chế độ không hồi tiếp

Bảng 2.4 TPS của các AES trong chế độ hồi tiếp

Thuật toán	Cấu trúc	Tối ưu	Slices	TPS
RC6	PP-2	Tốc độ	3189	**39662**
Rijndael	LU-2	Tốc độ	3528	**83387**
Serpent	LU-8	Tốc độ	7964	**55771**
Twofish	**SP-1-1**	**Area**	**2695**	**47380**



Hình 2.4 TPS của các AES trong chế độ không hồi tiếp

Theo các đánh giá trên thì:

- *Về thông lượng:* Serpent có thông lượng tốt nhất trong cả hai chế độ hồi tiếp và không hồi tiếp.
- *Về thông lượng trên slice:* Serpent có thông lượng tốt nhất trong chế độ không hồi tiếp; Rijndael có thông lượng tốt nhất trong chế độ hồi tiếp.

Kết luận của [18]

Về tốc độ thì Serpent và Rijndael ít nhất cũng nhanh gấp đôi 3 AES còn lại; Twofish và RC6 có tốc độ đạt mức trung bình và bằng nhau; Mars xếp hạng kém nhất. Serpent nhanh hơn Rijndael một chút nếu sử dụng cấu trúc Serpent I8 (8 vòng mã hóa coi là một vòng thực hiện), và chậm hơn đáng kể nếu sử dụng cấu trúc Serpent I1 (một vòng thực hiện giống như một vòng mã hóa). Twofish nhanh hơn RC6 từ 1% đến 54%.

Về thông lượng đối với đặc tính area, có thể chia các AES thành 2 nhóm chính. Rijndael và Serpent I8 có tốc độ cao nhất nhưng cần nhiều nhất các area. Twofish, RC6 và Serpent I1 hiệu quả nhất về area nhưng tốc độ chỉ trung bình. Mars kém nhất cả về tốc độ lẫn area.

Về khóa, đối với Rijndael, Serpent và Twofish, thì thời gian setup khóa chỉ chiếm một phần của thời gian cần thiết để mã hóa một khối data, và như thế có thể thay đổi khóa ngay khi đang làm việc mà không ảnh hưởng đến thông lượng mã hóa. Đối với Mars thì thời gian setup khóa lớn hơn thời gian mã một khối data, như thế khi đổi khóa cần thêm một khoảng thời gian, hoặc thêm mạch để nhớ khóa mới trong khi dùng nốt khóa cũ. Còn với RC6 thì kết luận của các nhóm không thống nhất.

Có thể thấy Serpent và Rijndael tốt hơn các thuật toán còn lại, trong đó Serpent nhỉnh hơn Rijndael. Tuy nhiên khi cân nhắc thêm các yếu tố khác NIST đã chọn Rijndael.

2.3.4 THỰC HIỆN MẬT MÃ TRÊN ĐƯỜNG ELLIPTIC BẰNG FPGA

Mật mã trên đường cong Elliptic (ECC) do Koblitz và Miller đề xuất vào năm 1985. So với các hệ mật khóa công khai khác như RSA và logarithm rời rạc thì ECC có độ dài khóa ngắn hơn, độ an toàn cao hơn. Đặc biệt ECC rất thích hợp với các ứng dụng nhúng vì việc cứng hóa cần ít tài nguyên hardware hơn. Ví dụ dùng VLSI để thực hiện ECC 155 bit chỉ cần 11,000 transistor, trong khi đó để có độ an toàn tương đương RSA phải 512 bit thực hiện trên 50,000 transistor [24].

ECC cũng an toàn hơn RSA. Để phá được ECC 97 bit cần công suất tính toán gấp đôi để phá RSA 512 bit.

Đã có nhiều nghiên cứu cả về lý thuyết lẫn thực hành để cứng hóa ECC trên FPGA: [24], [25] nghiên cứu ECC trong trường Galois hỗn hợp

$F_{(2^n)^m}$; [26] nghiên cứu bộ nhân super-serial... Khả năng tiêu thụ ít tài nguyên hardware của FCC là thuận lợi đáng kể cho việc cứng hóa FCC bằng FPGA so với AES.

2.3.5 THỰC HIỆN HÀM HASH BẰNG FPGA

Các hàm hash là cơ sở rất chung và quan trọng của mật mã học. Ứng dụng đầu tiên của chúng là sử dụng cùng hệ thống khóa công khai trong các sơ đồ chữ ký số. Chúng cũng là phần cơ bản của mã hóa xác thực thông báo (MAC). Các ứng dụng khác có thể kể ra là mã hóa nhanh, lưu trữ và kiểm tra mật khẩu, phát hiện vius máy tính, sinh số giả ngẫu nhiên...

Tuy thế rất khó thiết kế các hàm hash mạnh, không va chạm. Trong số hàng chục hàm hash được đề nghị thì chỉ có một số ít sử dụng được, số được chấp nhận là chuẩn còn ít hơn nữa.

SHA-1 là hàm hash được Mỹ chấp thuận làm chuẩn vào năm 1993, với kích thước 160 bit. Khi DES được thay bằng AES, thì độ an toàn của SHA-1 không đủ để tương thích với AES nữa. Do đó NSA đã phát triển 3 hàm hash mới tương ứng với AES 128, 192 và 256 bit khóa là SHA-256, SHA-384 và SHA-512.

Các hàm hash được thiết kế với định hướng cho software, tuy nhiên do những tính toán trong thực hiện hàm hash tương đối ít nên có thể dễ dàng thực hiện bằng hardware. So với AES và ECC thì cứng hóa hàm hash dễ hơn nhiều. Do hàm hash không đòi hỏi thông tin trạng thái nên có thể sử dụng mạch tổ hợp hay các vi xử lý công suất trung bình để thực hiện. Việc cứng hóa hàm hash bằng FPGA không sử dụng hết tài nguyên của một chip FPGA cỡ trung bình do thiết kế không cần sử dụng đến các thanh ghi và nhiều tài nguyên khác của FPGA.

So với SHA-1 thì SHA-256, -384 và -512 bit cần nhiều tài nguyên hardware hơn, nhưng tốc độ thực hiện lại nhanh hơn. Theo kết quả thực nghiệm trên cùng FPGA XCV-1000-6 của Xilinx [27] thì SHA-512 nhanh hơn SHA-1 33% đo bằng lý thuyết và 26% đo bằng thực nghiệm. Về tài nguyên thì SHA-512 sử dụng các mảnh CLB nhiều gấp đôi SHA-1, và cần thêm 4 Kbit RAM.

2.3.6 THỰC HIỆN SINH SỐ NGẪU NHIÊN BẰNG FPGA

Có ba kỹ thuật để tạo số ngẫu nhiên: đó là khuếch đại trực tiếp, chaos và trích mẫu dao động.

Khuếch đại trực tiếp là kỹ thuật quen thuộc, số ngẫu nhiên được lấy từ nguồn nhiễu nào đó (như từ nhiễu nhiệt của tiếp giáp bán dẫn). Nhược điểm của nguồn nhiễu nhiệt là rất kém ổn định và phụ thuộc nhiều vào môi trường.

Chaos là kỹ thuật tương tự kỹ thuật *Khuếch đại trực tiếp,* điểm khác là nguồn ngẫu nhiên dựa trên sự hỗn loạn nào đó từ một kích hoạt ban đầu.

Trích mẫu dao động sử dụng jitter của bộ dao động tần số thấp có hệ số phẩm chất Q thấp để trích mẫu nguồn tần số cao.

Hai kỹ thuật đầu liên quan đến xử lý tín hiệu tương tự đầu vào trước khi số hóa thành chuỗi bit ngẫu nhiên. Kỹ thuật thứ 3 hầu như chỉ xử lý số tín hiệu. Tại thời điểm này thì xử lý tín hiệu tương tự chưa phải là thế mạnh của FPGA. Bởi thế để thiết kế mạch tạo số ngẫu nhiên thì kỹ thuật *Trích mẫu dao động* thích hợp hơn cả.

Thiết kế mạch sinh số ngẫu nhiên không chiếm nhiều lắm tài nguyên của FPGA [33], [34] và có thể coi là dễ nếu so với thiết kế AES, khóa công khai hay elliptic.

2.4 AN TOÀN MẬT MÃ DỰA TRÊN HARDWARE

Các thiết kế mật mã trên cơ sở hardware an toàn hơn software rất nhiều. Tuy nhiên không phải cứ hardware là an toàn, mà độ an toàn có nhiều mức tùy thuộc vào thiết kế. Phần này sẽ phân tích toàn diện về độ an toàn của hardware.

Ở đây không xét đến các tấn công mã thám theo nghĩa chặn thu tín hiệu trên kênh truyền thông để thám khóa hay thám mã. Phần này chỉ luận bàn về các tấn công trực tiếp lên hardware, với giả thiết là kẻ tấn công tiếp cận được thiết bị và có cơ hội can thiệp sâu vào hardware cho mục đích tấn công thám ngược thiết kế.

2.4.1 TẤN CÔNG LÊN HARDWARE NÓI CHUNG

Các security memories hay security microcontroller đều được trang bị security bit. Khi bit này được lập trình thì không thể đọc ra được nội dung của chip. Tuy nhiên có thể xóa được các bit này bằng phương thức rất nghiệp dư [29], như làm biến động điện áp nguồn hay nhiệt độ. Ví dụ với microcontroller PIC16C84, mẹo được biết rộng rãi là nâng điện áp nguồn Vcc lên điệp áp nạp Vpp-0.5V trong lúc liên tục lặp lại thao tác ghi vào security bit sẽ xóa được security bit mà không ảnh hưởng đến phần còn lại của bộ nhớ.

Đối với security processor DS5000, sau một số lần gây sụt chớp nhoáng điện áp nguồn sẽ xóa được security bit mà dữ liệu trong bộ nhớ vẫn giữ nguyên. Các bộ xử lý như 8752 có thể được tấn công bằng cách hạ thấp điện áp nguồn để chuyển chế độ làm việc giữa bộ nhớ trong và bộ nhớ ngoài, nhờ đó đọc ra được nội dung bộ nhớ. Hạ thấp điện áp nguồn còn áp dụng trong trường hợp, ví dụ đối với bộ sinh số ngẫu nhiên analogue, khi điện áp nguồn giảm nhẹ, sẽ sinh ra hầu như toàn số ‘1’.

Với smartcard, cách đơn giản là cậy lớp eboxi bao phủ trên chip, tiếp theo dùng axit cho ăn mòn lớp bảo vệ chip đến khi lộ ra mặt silicon với các đường dẫn bus. Từ đó cho smartcard làm việc bình thường rồi dùng các đầu dò nhỏ thăm vào bus để đọc ra số liệu. Tuy nhiên đây là kiểu làm amateur.

Trong các phòng thí nghiệm thiết bị bán dẫn người ta sử dụng kỹ thuật khác để thám ngược thiết kế trong chip [30]. Đó là ăn mòn từng lớp một của chip để lộ ra các lớp N và P bằng cách sử dụng hiệu ứng Schottky: cho lắng đọng trên chip một lớp mỏng kim loại như vàng hay palladium để tạo ra diot. Dùng chùm tia điện tử để chụp ảnh diot này. Ảnh của các lớp liên tiếp của chip qua phần mềm xử lý ảnh máy tính, sẽ tái tạo từ ảnh mờ thành bản rõ nét các đường. Kết quả là được một sơ đồ mặt nạ chip từ đó cho phép nhận dạng thiết kế, hay thậm chí được một thư viện các cell từ đó có thể tạo lại chip. Với cách làm này thì thám ngược vi xử lý 80386 của Intel mất thời gian là 2 tuần và chi phí khoảng 6 con chip.

Khi đã biết sơ đồ bố trí layout và chức năng của chip, thì có một kỹ thuật cực mạnh do IBM phát triển để quan sát hoạt động của nó mà không cần gỡ bỏ các lớp oxi hóa: đặt một tinh thể chất lithium niobate lên điểm tại đó cần theo dõi điện áp. Chỉ số khúc xạ của chất này thay đổi khi áp vào điện trường, và điện thế của silicon nằm dưới có thể đọc ra bằng cách sử dụng tia laser cực tím xuyên lướt qua tinh thể. Đây được hiểu là cách chuẩn để phục hồi khóa mã từ các chip đã biết sơ đồ bố trí layout [29].

Để chống lại tấn công kiểu này, quân đội Mỹ sử dụng một chất phủ chip [1]. Chất này không chỉ chắn ánh sáng và dẫn điện, mà còn chống lại sự gỡ bỏ nó: những toan tính gỡ bỏ sẽ làm hỏng luôn chất silicon phía dưới. Không chỉ để che phủ chip, chất này còn có khả năng gây nhầm lẫn

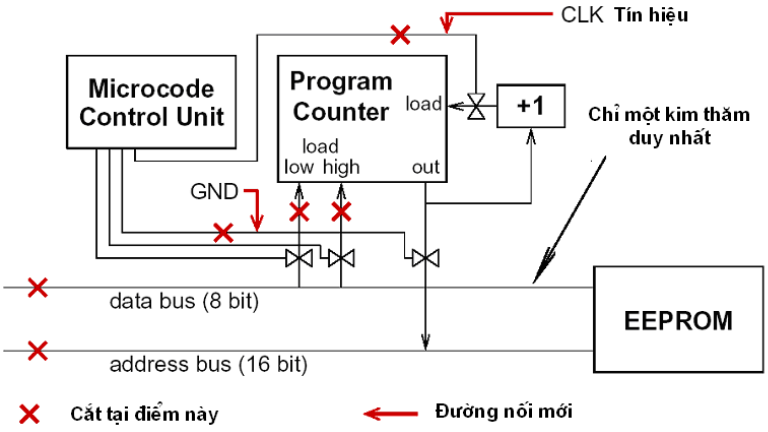
cho người tấn công. Chẳng hạn thấy dường như phần tử thiết kế là transistor, nhưng thực ra đó chỉ là đường nối giữa cổng và nguồn điện; hay cổng NOR 3 đầu vào thì thấy như là NOR 2 đầu vào.

Một giải pháp mang tính hệ thống hơn đã được sử dụng trong chip Clipper của chính phủ Mỹ. Chip có một hệ thống đường dẫn kiểu cầu chì, các đường dẫn tạo ra một thuật toán mã hóa cùng một khóa dài hạn cũng theo kiểu cầu chì tạo nên silicon vô định hình gây khó khăn cho việc soi hiển vi chip. Thêm nữa, bề mặt của chip được "ướp muối" bằng các bộ dao động để làm khó cho tấn công kiểu cảm biến điện từ. Tuy nhiên bằng kỹ thuật cắt lớp với kính hiển vi điện tử, một đội thuộc Cambridge đã thông báo thám được thiết kế một chip Clipper (cũng cần chú thích là việc thám này dựa vào lỗi của giao thức chứ không phải bằng cách thâm nhập vật lý).

Cắt lớp không chỉ để thám thiết kế các chip có bảo vệ bề mặt. Phòng thí nghiệm Sandia National đã phát minh kỹ thuật nhìn xuyên qua chip từ phía sau bằng laser hồng ngoại tại bước sóng ở đó chất silicon trở nên trong suốt. Khi ấy dòng quang điện tạo ra cho phép dò hoạt động của chip và nhận dạng trạng thái logic của từng transistror một.

Để tấn công smartcard, có những thiết bị chuyên dụng tạo chùm tia ion FIB. Thiết bị này có thể cắt các rãnh trong lớp kim loại của chip và tạo rãnh mới hoặc các lớp cách điện. Nó cũng có thể cấy ion vào để thay đổi lớp ngoài của silicon và thậm chí có thể tạo các lỗ qua cấu trúc dẫn điện của lớp thấp nhất của chip. Thiết bị này có giá khoảng vài triệu U.S dollar và cũng có thể thuê ở một số công ty bán dẫn.

Với thiết bị này việc tấn công smartcard trở nên đơn giản. Cách tấn công điển hình là ngắt hầu hết các chân của CPU ra khỏi bus, chỉ để lại các chân liên quan đến việc đọc EEPROM như Hình 2.5 dưới đây.



Hình 2.5 Tấn công smartcard

Khi ấy thâm nhập bộ nhớ không phải do bộ đếm chương trình điều khiển mà từ đường tín hiệu clock và người tấn công chỉ cần một kim thăm hoặc một đầu dò quang điện là đọc được toàn bộ nội dung EEPROM.

2.4.2 TẤN CÔNG LÊN FPGA

Phần trên đã lướt qua một vài nguy cơ và kỹ thuật tấn công lên thiết kế hardware. Với FPGA có thể chia các kiểu tấn công thành các nhóm [28] như sau.

2.4.2.1 TẤN CÔNG KIỂU HỘP ĐEN

Tấn công kiểu hộp đen là phương pháp cổ điển để thám ngược thiết kế trong một con chip. Kẻ tấn công cho vào mọi tổ hợp có thể và ghi lại các tín hiệu ra tương ứng, từ đó cố gắng suy luận ra logic bên trong của FPGA nhờ bìa Karnaugh hay thuật toán nào đó. Tấn công này chỉ hiệu quả nếu FPGA nhỏ và các đầu vào, ra là rõ ràng, cộng với sức mạnh của nhiều máy tính. Khi kích thước và độ phức tạp của FPGA tăng thì mức độ thành công của tấn công kiểu này càng giảm.

2.4.2.2 TẤN CÔNG KIỂU ĐỌC LẠI

Đọc lại là đặc điểm của hầu hết các họ FPGA, cho phép đọc ra cấu hình của FPGA để debug. Ý tưởng tấn công là đọc ra cấu hình qua cổng

JTAG hoặc qua giao diện lập trình để từ đó tìm được khóa hay các thông tin mật. Chức năng này có thể cấm được nhờ các bit security - đây là tính năng khóa do nhà máy sản xuất chip cung cấp và đã được cấp bằng sáng chế.

Tuy nhiên, có các biện pháp đơn giản để vượt qua rào cản này. Bằng các kỹ thuật gây lỗi lên phần cứng, chẳng hạn như phát xạ điện từ, laser hồng ngoại, hay thậm chí đèn chớp flash. Hầu như luôn có thể phá được các bit khóa và đọc ra được cấu hình của FPGA. (Tuy nhiên với ASIC thì lại là vấn đề khác).

2.4.2.3 TẤN CÔNG NHÁI LẠI SRAM FPGA

FPGA công nghệ SRAM sử dụng bộ nhớ ngoài kiểu PROM để lưu trữ cấu hình. Thường thì bộ nhớ ngoài không được bảo vệ hoặc bảo vệ đơn giản bằng security bit và có thể phá được như cách đã trình bày ở phần trên. Hoặc tấn công theo cách khác. Tại thời điểm bật nguồn thông tin trong bộ nhớ được truyền đến FPGA để cấu hình nó. Người tấn công có thể móc que thăm trên đường truyền và thu được toàn bộ thông tin cấu hình này.

2.4.2.4 THÁM NGƯỢC THIẾT KẾ TỪ CHUỖI BIT

Các tấn công lên FPGA đã nói ở trên đều cho kết quả là chuỗi bit mô tả thiết kế của chip. Người tấn công sẽ từ chuỗi bit này phân tích ra thuật toán mã hóa hoặc khóa mật.

Mặc dù các công ty sản xuất FPGA tuyên bố là độ an toàn của chuỗi bit dựa trên cấu trúc của số liệu cấu hình. Nhưng cấu trúc này chỉ bí mật khi thỏa thuận không công bố được ký kết. Điều này, xét trên quan điểm mật mã, thì coi như không mật. Và thực tế hơn 10 năm trước công ty phần mềm CAD NEOCad đã phá được và thám ra thiết kế FPGA của Xilinx. Họ đã tạo lại thông tin cần thiết như các bảng look-up, đường nối, các

phần tử lưu trữ. Từ đó NEOCad sản xuất phần mềm thiết kế mà không cần ký kết hiệp định không công bố với hãng sản xuất FPGA. Đối với các tổ chức tấn công lớn cấp chính phủ, cũng có thể họ nhận thông tin trực tiếp từ người bán hay từ các công ty đã ký kết hiệp định không công bố.

2.4.2.5 TẤN CÔNG VẬT LÝ

Tấn công vật lý là thăm dò các điểm trong chip để nghiên cứu thiết kế chip từ đó tìm thông tin về thuật toán hay khóa mật. Tấn công này nhằm vào các bộ phận trong chip không thể tiếp cận thông qua các chân I/O thông thường. Bằng mất thường hoàn toàn có thể làm được điều này với kính hiển vi quang học và đầu dò cơ khí đối với các FPGA đơn giản. Với FPGA phức tạp cần phương pháp tiên tiến hơn, như sử dụng hệ thống chùm tia ion FIB.

Nói chung không có biện pháp nào để bảo vệ FPGA chống lại được dạng thức tấn công vật lý. Tiếp theo chúng ta sẽ phân tích những nỗ lực cần thiết để tấn công vật lý lên các FPGA được sản xuất bằng các công nghệ khác nhau.

SRAM FPGAS: Tuy có rất ít công bố về tấn công vật lý lên SRAM FPGA, nhưng những nghiên cứu về tấn công loại này lên bộ nhớ SRAM lại rất nhiều cả về học thuật lẫn thực nghiệm. Do cấu trúc bộ nhớ SRAM cũng tương tự như của SRAM trong FPGA nên có thể thấy là tấn công lên chúng cũng cùng kiểu. "I_{DDQ} testing" là phương pháp được sử dụng rộng rãi nhất. Phương pháp này dựa trên việc phân tích dòng điện, bằng cách thực hiện một loạt các vector test cho đến khi tìm được điểm tại đó đo được dòng điện. Sau đó căn cứ vào đặc tính I_{DDQ} bất bình thường giữa các trạng thái khác nhau để xác định các thông số cần thiết. Một khả năng tấn công khác là dựa vào đường dẫn scan trong chip, đường dẫn này do các nhà sản xuất chip thêm vào cho mục đích test chip.

Để thâm nhập đến các điểm không được nhà máy dành sẵn, thì phải gỡ bỏ các lớp của chip. Cách truyền thống là thăm cơ học bằng đầu dò vonfram đường kính 0.1 ÷ 0.2 μm. Đầu thăm này cung cấp băng thông hàng gigahert với tụ 100 fF và điện trở 1M. Tuy nhiên nếu chip có cấu trúc phức tạp và nhiều lớp thì phương pháp cơ học này không đủ. Khi ấy lại cần đến thiết bị chùm tia ion FIB. Thiết bị có vai trò tương tự như kính hiển vi điện tử, có thể quan sát những cấu trúc xuống đến dưới 5 nm để tìm ra các chất dẫn điện được che dấu và tạo các điểm thăm mới.

Một kiểu khác là dùng thiết bị test theo kiểu tia điện tử EBT. EBT là kính hiển vi điện tử đặc biệt có thể tăng tốc các điện tử sơ cấp lên đến 2.5 kV tại 5nA. EBT đo năng lượng và số lượng điện tử thứ cấp phát xạ ra.

Tấn công vật lý lên SRAM FPGA là có thể, tuy nhiên giá thành rất cao và hầu như chỉ có thể thực hiện được với các tổ chức lớn như cơ quan tình báo.

ANTIFUSE FPGAS: cấu trúc cơ bản của một antifuse node (AF) là một lớp cách điện mỏng (nhỏ hơn $1\mu m^2$) nằm giữa các chất dẫn điện. Lập trình bằng cách áp điện áp vào các chất dẫn này. Khi ấy chất cách điện trở nên có điện trở thấp và sinh ra một đường nối (đường kính khoảng 100 nm) giữa các chất dẫn. Trạng thái này tồn tại vĩnh viễn.

Để phát hiện được đường nối này có tồn tại hay không cần phải gỡ bỏ từng lớp một và/hoặc sử dụng phương pháp cắt lớp. Tuy vậy chưa thấy công bố thành công nào về tấn công kiểu này. Để phân tích tìm cấu hình của một cell, cần có nhiều phép thử. Khó khăn chính là lớp cách điện quá nhỏ so với một cell AF. Quá trình thử trên một cell sẽ gây lỗi và dường như những cell còn lại sẽ bị phá hủy [31]. Để thám ngược thiết kế một file cấu hình của chip Actel A54SX16 có 24.000 cổng hệ thống cần tiêu tốn

chừng 800.000 con chip có cùng cấu hình. Khó khăn nữa cho người tấn công là chỉ có khoảng 2-5% số đường nối có thể trong một thiết kế trung bình được sử dụng thực sự. Bởi vậy tấn công Antifuse FPGA rất khó, khó và tốn kém hơn cả đối với ASIC. Thực tế tấn công AF FPGA nhằm thay thế một cell cần 2 tháng với chi phí 1000 $ [32].

FLASH FPGAS: Các đường nối trong flash FPGA được thực hiện bằng flash transistor, có nghĩa là tổng số điện tử chảy qua cổng thay đổi sau khi cấu hình và không có khác biệt về quang học như trường hợp AF FPGA. Flash FPGA có thể được phân tích bằng cách đặt chip trong buồng chân không rồi cấp nguồn cho nó. Sau đó dùng kính hiển vi điện tử thứ cấp để quan sát sự chuyển động của điện tử. Cách tấn công này đòi hỏi người tấn công phải gỡ bỏ các gói để thâm nhập đến vết khắc trên silicon. Tuy nhiên kiểu này phức tạp và những nhà chuyên môn hiện vẫn đang tranh cãi về tính thực tiễn của nó.

Ngoài ra còn có thể tấn công lên flash FPGA nhằm vào vùng liên quan của bộ nhớ flash. Cách tấn công này cũng tương tự như tấn công lên EEPROM.

2.4.2.6 TẤN CÔNG SIDE CHANNEL

Một thiết bị bất kỳ khi làm việc đều bộc lộ thông tin về mình dưới một hình thức nào đó. Các bộc lộ đó hiểu như một kênh thông tin phụ hay là *side channel*, gồm nhiều dạng: sự tiêu thụ năng lượng, kiểu hoạt động theo thời gian, phát xạ điện từ. Tấn công dựa vào side channel gọi là tấn công side channel.

Có hai kiểu tấn công side channel chính: Phân tích năng lượng đơn giản (SPA) và Phân tích năng lượng vi sai (DPA). Ở đây người tấn công phân tích tiêu thụ năng lượng của thiết bị trong khi thực hiện một thao tác

mật mã nhằm tìm ra khóa mật mà không phải thấm nhập vào thiết bị. Ý tưởng chính của DPA là tìm các khu vực ở đó tiêu thụ năng lượng có liên quan đến khóa mật. Cách tấn công này, trong một số trường hợp, thành công ngay cả khi biết rất ít hoặc không biết gì về đối tượng. Do đó đã có nhiều nghiên cứu để hoàn thiện kiểu tấn công này [28].

Có thể kết luận gì về các tấn công lên FPGA?

Với tấn công kiểu hộp đen: tấn công này chỉ hiện thực với "hộp" đơn giản. Ngày nay, sự phức tạp của thuật toán mật mã cộng với sự phức tạp của cấu trúc FPGA tự bản thân nó đã chống lại hiệu quả tấn công kiểu này.

Tấn công kiểu nhái lại SRAM FPGA: khó mà chống lại hiệu quả tấn công này. Các giải pháp được đề nghị nhằm chống lại móc trộm thông tin trên đường dẫn dữ liệu từ bộ nhớ ngoài đến FPGA đều không hiện thực hoặc lại nảy sinh vấn đề phức tạp khác. Giải pháp "thực tế" hơn cả là chế tạo một chip chứa cả bộ nhớ bất biến lẫn FPGA, hoặc xếp hai chip cạnh nhau rồi đổ epoxy phủ lên cả đôi.

Biện pháp đối phó hiệu quả và thực tế nhất chống tấn công nhái lại SRAM FPGA là mã hóa file cấu hình hoặc mã một phần chuỗi bit. Đã có nhiều patent và các công bố nghiên cứu về việc này [28]. Họ FPGA 60RS của Actel chứa khóa trên chính chip FPGA để giải mã file cấu hình. Tuy nhiên hãng chỉ có một khóa cho tất cả các chip, có nghĩa nếu lộ khóa thì sẽ lộ file cấu hình của tất cả các thiết kế liên quan đến chip FPGA cùng họ này. Một giải pháp khác là nuôi toàn bộ SRAM FPGA bằng pin và như vậy không cần truyền file cấu hình mỗi khi bật nguồn. Tuy nhiên giải pháp này không kinh tế vì cần pin cho FPGA. Bởi vậy hiệu quả hơn cả là tổ hợp cả hai giải pháp mã hóa và nuôi bằng pin: FPGA Virtex II của Xilinx có khối giải mã 3DES trên chip với hai khóa lưu trong vùng RAM

được nuôi bằng pin.

Với tấn công vật lý: đối với SRAM FPGA thì cần nhất là hiệu ứng lưu trạng thái của các cell càng nhỏ càng tốt. Trong điều kiện nhiệt độ bình thường trong phòng (20^0C) thì SRAM cell sau khi ngắt nguồn nuôi vẫn có thể lưu được trạng thái đến hàng tháng, và ở 0^0C thì hàng năm. Đối phó hiệu ứng này bằng cách định kỳ đảo hoặc di chuyển dữ liệu quanh bộ nhớ.

Antifuse FPGA có lẽ là công nghệ tự nó an toàn nhất với tấn công vật lý: phần trên ta đã biết mỗi toan tính tấn công lên một cell của FPGA loại này đều có thể làm hỏng luôn các cell còn lại của chip. Không thấy công bố nào đề xuất giải pháp bảo vệ Antifuse FPGA trước tấn công vật lý, ngoài những khuyến cáo chung chung về bảo vệ môi trường quanh chip.

Đối với các ô nhớ flash/EEROM, những lần nạp đầu tiên tạo ra sự dịch chuyển lớn trong ngưỡng của cell. Hiệu ứng này sẽ giảm sau khoảng 10 lần ghi/xóa. Bởi vậy để loại trừ hiệu ứng này nên nạp FPGA khoảng 100 lần bằng dữ liệu ngẫu nhiên.

Với tấn công đọc lại: có thể sử dụng security bit để chống lại tấn công này. Để đối phó với thủ pháp gây lỗi nhằm xóa security bit, phải đặt vào môi trường an toàn ở đó có cơ chế phát hiện can thiệp để xóa nội dung của FPGA hoặc thậm chí hủy cả chip.

Với tấn công Side Channel: có thể chia các biện pháp đối phó thành hai loại: bằng software và bằng hardware. Giải pháp software như là che khóa mật bằng giá trị ngẫu nhiên. Biện pháp phần cứng có thể là gây nhiễu xóa bức xạ hoặc thay đổi mức transistor của logic.

Tóm lại*, trong phần này chúng ta đã xem xét qua một lượt các công nghệ*

hiện đang phổ biến. Những phân tích dẫn đến công nghệ thích hợp nhất cho mật mã là các bộ xử lý có khả năng cấu hình lại mà đại diện là FPGA. Chúng ta cũng xem xét khả năng của FPGA khi thực hiện các bài toán cơ bản của mật mã; vấn đề security của FPGA trước các tấn công cũng được nghiên cứu.

PHẦN 3.
CHUẨN BỊ ĐỂ CỨNG HÓA MẬT MÃ

Hai phần trước đã đưa chúng ta đến lựa chọn FPGA cho mật mã. Bởi vậy các kiến thức cần thiết để cứng hóa mật mã cũng xoay quanh FPGA. Một số kiến thức về toán thích hợp cho thiế kế ứng dụng nhúng cũng được đề nghị.

3.1 CÁC KIẾN THỨC CẦN THIẾT ĐỂ THỰC HIỆN FPGA

3.1.1 KIẾN THỨC VỀ TOÁN

Ở đây chỉ liệt kê các kiến thức liên quan trực tiếp đến quá trình cứng hóa thuật toán, theo hiểu biết của người làm kỹ thuật. Vấn đề này có thể còn phải thảo luận thêm nhiều trong quá trình nghiên cứu chuyên sâu sau này, với sự tham gia của những người làm khoa học mật mã. Các kiến thức về toán cần thiết bao gồm:

- Thuật toán bình phương và nhân: thuật toán thông dụng nhất cho phép lũy thừa modulo.
- Thuật toán nhân modulo Montgomery: thích hợp đối với các ứng dụng nhúng ở đó giới hạn về dung lượng bộ nhớ.
- Các phép toán trên trường F_{2^n} .
- Các phép toán trên đường cong elliptic.

3.1.2 KIẾN THỨC VỀ KỸ THUẬT

Kiến thức để cứng hóa đương nhiên phải là mọi kiến thức về điện tử cộng với kiến thức về lập trình. Cũng cần chú thích thêm là lập trình cho chip, cho dù bằng ngôn ngữ của các bộ xử lý hay bằng ngôn ngữ mô tả

phần cứng của FPGA, cũng phải theo tư duy kiểu kỹ thuật, có nghĩa các lệnh gắn trực tiếp với các thanh ghi, các cổng, các xung... Các nhà phát triển phần mềm thường có khuynh hướng tư duy nối tiếp, cụ thể bằng các dòng lệnh nối tiếp nhau. Ngay cho dù trong một ứng dụng đa nhiệm thì thực hiện mỗi lệnh cũng do chỉ một máy. Các nhà phát triển phần cứng lại có thiên hướng tư duy và lập trình kiểu song song, bởi một mạch điện thường có nhiều đầu vào hoạt động đồng thời trên nhiều macrocell liên kết với nhau cho ra một hoặc nhiều tín hiệu ra đồng thời. Do đó phát biểu trong ngôn ngữ mô tả phần cứng tạo ra cấu trúc. Hiệu quả của lập trình nhiều khi phụ thuộc vào cách chương trình phối hợp với hoạt động của phần cứng hơn là theo thuật toán.

Các kiến thức căn bản về điện tử có thể kể ra là: Lý thuyết mạch, Đại số Boolean, Nguyên lý kỹ thuật điện tử, Kỹ thuật xung, Kỹ thuật số, Mạch tích hợp, Điện tử tương tự và điện tử số, Kỹ thuật vi xử lý, Linh kiện bán dẫn và vi mạch, Kỹ thuật đo lường điện tử, Tín hiệu nhỏ, Ngôn ngữ lập trình cho vi xử lý assambler, C, ngôn ngữ mô tả phần cứng VHDL, Verilog, ABEL...

3.1.3 KIẾN THỨC VỀ CÔNG NGHỆ

Thiết kế nguyên lý đúng chưa chắc đã dẫn đến thành công khi chế tạo, đặc biệt với những mạch làm việc ở tần số cao và siêu cao. Cần kiến thức về Tương thích điện từ, kỹ thuật Thiết kế mạch in và môi trường làm việc (nhiệt độ, độ ẩm, khả năng chịu rung xóc...) để bố trí, sắp xếp các linh kiện sao cho tác động nhiễu điện từ và nhiễu nhiệt cũng như các ảnh hưởng qua lại của chúng ít nhất hoặc bù được cho nhau.

3.1.4 KIẾN THỨC VỀ CÔNG NGHỆ VÀ THỊ TRƯỜNG VI MẠCH

Am tường về các chủng loại vi mạch hiện có giúp người thiết kế chủ động, lựa chọn tối ưu nhất linh kiện phù hợp cho bài toán xác định.

Kiến thức về phần này phải được cập nhật thường xuyên do của công nghệ chế tạo chip luôn phát triển và thị trường chip luôn biến động. Tại thời điểm này một chip FPGA chứa 500,000 cổng, và số lượng này gấp đôi cứ sau mỗi 18 tháng, đồng thời giá chip cũng ngày một giảm. Cũng cần đánh giá được cùng một chủng loại chip thì của hãng nào có chất lượng cao hơn, của hãng nào rẻ hơn; công nghệ chip nào có tương lai và công nghệ nào đang thoái hóa... Hoạt động này tương tự như thu thập thông tin tư liệu góp phần quyết định vào lựa chọn giải pháp đúng đắn cho thiết kế hiện tại và hướng đi của tương lai.

3.2 CÔNG CỤ CẦN THIẾT ĐỂ THỰC HIỆN FPGA

3.2.1 CÔNG CỤ THIẾT KẾ

Quá trình thiết kế FPGA sẽ qua các bước điển hình sau. Mỗi tác vụ cần một software tương ứng. Công cụ để thiết kế FPGA là bộ phần mềm Computer-Aided Design (CAD) của chính hãng hay do các hãng thuộc thành phần thứ ba sản xuất.

- Nhập thiết kế: vẽ sơ đồ nguyên lý bằng công cụ đồ họa của CAD hay mô tả bằng ngôn ngữ mô tả phần cứng, hay phối hợp cả hai
- Tối ưu logic: thuật toán nhằm làm cho thiết kế đạt hiệu quả cao nhất có thể về tài nguyên, tốc độ ứng với chip sẽ sử dụng.
- Technology mapper: ánh xạ từ các cổng logic cơ bản vào các khối logic của FPGA.
- Placement: chọn lựa khối logic nào sẽ sử dụng.
- Router: sử dụng tài nguyên đường dẫn để nối các khối với nhau.
- Mô phỏng: để kiểm tra tính đúng đắn của hoạt động của chip. Trong quá trình này người thiết kế phải tìm lỗi và quay lại bước

đầu để sửa.

- Nạp vào chip: khi quá trình mô phỏng đã hoàn tất, logic theo đúng ý tưởng thì bước cuối cùng là nạp thiết kế vào chip bằng công cụ nạp

Bộ công cụ lớn, mạnh và nhiều tính năng nhất thuộc về Altera và Xilinx. Ngoài ra Synplicity là một trong các công ty phần mềm khác phát triển công cụ cho FPGA.

3.2.2 THIẾT BỊ

- *Máy tính:* để chạy các phần mềm CAD cho quá trình thiết kế FPGA. Một thiết kế FPGA thường phức tạp, thời gian chạy mô phỏng và gỡ rối một thiết kế có độ phức tạp trung bình mất từ vài giờ đến vài ngày tùy thuộc cấu hình máy tính. Một điều quan trọng là màn hình càng lớn càng thuận lợi cho người thiết kế vì các chi tiết trong FPGA (các khối logic, các đường dẫn...) rất nhiều và nhỏ.

- *Thiết bị nạp:* để nạp chuỗi bit (kết quả kết xuất của phần mềm mô phỏng) vào chip. Ngày nay đa số các FPGA đều có khả năng tự nạp mà không cần thiết bị chuyên dụng. Trong trường hợp ấy truyền trực tiếp chuỗi bit từ máy tính vào chip qua cổng JTAG.

3.2.3 NHÂN LỰC

Hầu hết các bước thiết kế cần trình độ kỹ sư chuyên ngành điện tử am hiểu cả về phân tích, thiết kế mạch lẫn lập trình. Cũng cần chú ý đến phẩm chất của người làm nghiên cứu: tỉ mỉ, cẩn thận và say mê.

3.3 CÁC HÃNG SẢN XUẤT FPGA

Hai hãng chính sản xuất FPGA là Xilinx và Altera. Các hãng Actel,

QuickLogic, Lattice, Cypress, AT&T, Atmel, ICT, Lucent Technologies, Rohm, Space Electronics chiếm thị phần còn lại. FPGA của các hãng này thường có thêm chức năng xác định.

Hai loại FPGA chính đang có trên thị trường: SRAM FPGA và Antifuse FPGA. Loại thứ nhất thì các hãng Xilinx và Altera đang dẫn đầu, ngoài ra hãng AT&T cũng đang là đối thủ. Loại thứ hai thì Actel, Quick-logic và Cypress, và Xilinx đang cạnh tranh nhau.

SRAM FPGA phổ biến hơn Antifuse FPGA. Tuy nhiên trong các lĩnh vực ở đó vấn đề bảo vệ chip chống thâm nhập là quan trọng thì Antifuse FPGA được ưa chuộng hơn.

3.4 TƯƠNG LAI CỦA FPGA

Trước kia FPGA chỉ gồm các cổng logic được nối với nhau để thực hiện một thiết kế nào đó. Hiện nay các nhà sản xuất FPGA đang có khuynh hướng kết hợp thêm các phần xác định vào cùng một chip FPGA dưới dạng nhân cứng hay nhân mềm, như giao diện I/O PCI, giao diện mạng hay vi điều khiển, vi xử lý RISC, DSP.

Vai trò của FPGA đang thay đổi. Nó không đơn thuần là một loại chip để cứng hóa một thuật toán mà trở thành platform trên đó tổ hợp nhân cứng, nhân mềm, logic có thể lập trình. Trong tương lai gần trên một chip FPGA sẽ có tất cả các thành phần cần thiết để tạo nên System-on-Chip. Chip mới sẽ có tốc độ của ASIC và tính mềm dẻo của FPGA.

***Tóm lại**, phần này đã đề xuất những kiến thức chúng tôi cho là cần thiết trong thiết kế điện tử nói chung và FPGA nói riêng để cứng hóa mật mã. Những đề nghị này tuy dựa trên cơ sở nghiên cứu của hai phần trước, nhưng vẫn là chủ quan theo nhận thức của người viết. Thực tế quá trình cứng hóa sẽ bổ sung đầy đủ thêm.*

KẾT LUẬN

1. Không thể khẳng định hardware hay software cái nào hơn cái nào trong mật mã, tuy nhiên có thể khẳng định ưu thế vượt trội của hardware là tốc độ và an toàn chống xâm nhập (và kèm theo là giá thành!).
2. Trong thế giới hardware thì công nghệ FPGA là thích hợp nhất cho mật mã theo cả 2 nghĩa *encryption* và *security*.
3. Quá trình làm chủ FPGA tốn nhiều thời gian và công sức. Nhiều công ty nhỏ trên thế giới chọn giải pháp tình báo công nghiệp để nhanh chóng có sản phẩm.
4. Tương lai của FPGA không dừng lại ở khả năng cứng hóa một tác vụ nào đó, mà ở chỗ FPGA đang tiến đến System-on-Chip.

TÀI LIỆU THAM KHẢO

1. FIPS 140-1 - *Security Requirements for Cryptographic Modules*., 1994 January 11.

2. Leon Adams., *Choosing the Right Architecture for Real-Time Signal Processing Designs*., White Paper., SPRA879 - November 2002.

3. Christof Paar., *Reconfigurable Hardware in Modern Cryptography*., ECC 2000 October 4-6., Essen, Germany.

4. Hagai Bar-El., *Security Implications of Hardware vs. Software Cryptographic Modules*., Information Security Analyst., October 2002.

5. Cryptology., http://www.cyphernet.org/cyphernomicon/5.html

6. Leon Adams., *Choosing the Right Architecture for Real-Time Signal Processing Designs*., SPRA879 - November 2002

7. Stephen Brown and Jonathan Rose., *Architecture of FPGAs and CPLDs: A Tutorial.,* Department of Electrical and Computer Engineering University of Toronto.

8. Khary Alexander, Ramesh Karri, Igor Minkin, Kaijie Wu, Piyush Mishra, Xuan Li., *Towards 10-100 Gbps Cryptographic Architectures*., IBM Corporation, Poughkeepsie, NY, 12601.

9. AJ Elbirt, C Paar., *Towards an FPGA Architecture Optimized for Public-Key Algorithms.,* Cryptography and Information Security Laboratory, Worcester, MA 01609.

10. Thomas Blum., *Modular Exponentiation on Reconfigurable*

Hardware., Thesis., WORCESTER POLYTECHNIC INSTITUTE.

11. M. Shand and J. Vuillemin. *Fast implementations of RSA cryptography*. In Proceedings 11th IEEE Symposium on Computer Arithmetic, pages 252–259, 1993.

12. H.Orup. *Simplifying quotient determination in high-radix modular multiplication*., In Proceedings 12th Symposium on Computer Arithmetic, pages 193–9, 1995.

13. K. Iwamura, T. Matsumoto, and H. Imai. *Montgomery modular-multiplication., method and systolic arrays suitable for modular exponentiation*. Electronics and Communications in Japan, Part 3, 77(3):40–51, March 1994.

14. J.-P. Kaps. *High speed FPGA architectures for the Data Encryption Standard*., Master's thesis, ECE Dept., Worcester Polytechnic Institute, Worcester, USA, May 1998.

15. Ahmed Shihab, Alcahest; and Martin Langhammer, Altera., *Implementing IKE Capabilities in FPGA Designs*., Dec 05, 2003 URL: http://www.commsdesign.com/showArticle.jhtml?articleID=16600061

16. Alexander Tiountchik, Institute of Mathematics, National Academy of Sciences of Belarus và Elena Trichina, Advanced Computing Research Centre, University of South Australia., *FPGA Implementation of Modular Exponentiation*.

17. Hauck, S. (1998). "*The Roles of FPGAs in Reprogrammable Systems*" Proceedings of the IEEE 86(4): 615-638.

18. Kris Gaj and Pawel Chodowiec., *Hardware performance of the*

AES finalists - survey and analysis of results., George Mason University.

19. AJ Elbirt, W Yip, B Chetwynd, C Paar., *An FPGA-Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists*., ECE Department, Worcester Polytechnic Institute.

20. Kris Gaj and Pawel Chodowiec., *Comparison of the hardware performance of the AES candidates using reconfigurable hardware.,* George Mason University.

21. Bruce Schneier, John Kelseyy, Doug Whitingz, David Wagnerx, Chris Hall, Niels Ferguson., *Performance Comparison of the AES Submissions.,* January 3, 1999.

22. J. P. Kaps and C. Paar, *Fast DES implementation on FPGAs and its application to a universal key-search machine*, in Fifth Annual Workshop on Selected Areas in Cryptography, vol. LNCS 1556, Springer-Verlag, August 1998.

23. O. Mencer, M. Morf, and M. J. Flynn, *Hardware Software Tri-Design of Encryption for Mobile Communication Units*, in Proceedings of International Conference on Acoustics, Speech, and Signal Processing, vol. 5, (New York, New York, USA).

24. K. H. Leung, K. W. Ma, W. K. Wong và P. H. W. Leong., *FPGA Implementation of a Microcoded Elliptic Curve Cryptographic Processor.,* Department of Computer Science and Engineering, The Chinese University of Hong Kong.

25. *M. Rosner Elliptic Curve Cryptosystems on reconfigurable hard-*

ware., Master's Thesis Worcester., Polytechnic Institute Worcester USA 1998.

26. G. Orlando and C. Paar., *A super-serial Galois field multiplier for FPGAs and its application to public key algorithms.,* Proceedings of the IEEE Symposium on Field-programmable custom computing machines., trang 232-239., 1999.

27. T. Grembowski, R. Lien, K. Gaj, N. Nguyen, P. Bellows, J. Flidr, T. Lehman, B. Schott., *Comparative Analysis of the Hardware Implementations of Hash Functions SHA-1 and SHA-512.*, Electrical and Computer Engineering, George Mason University, 4400 University Drive, University of Southern California - Information Sciences Institute.

28. Thomas Wollinger and Christof Paar., *How Secure Are FPGAs in Cryptographic Applications?.*,

    Report 2003/119, http://eprint.iacr.org/, 5. June 2003

29. Ross Anderson Markus Kuhn., *Tamper Resistance - a Cautionary Note.*, The Second USENIX Workshop on Electronic Commerce Proceedings, Oakland, California, November 18-21, 1996, pp 1-11, ISBN 1-880446-83-9.

30. S Blythe, B Fraboni, S Lall, H Ahmed, U deRiu, *Layout Reconstruction of Complex Silicon Chips*, IEEE Journal of Solid-State Circuits v 28 no 2 (Feb 93) pp 138-145.

31. B. Dipert. *Cunning circuits confound crooks.,* http://www.einsite.net/ednmag/contents/images/21df2.pdf.

32. G. Richard., *Digital Signature Technology Aids IP Protection*., EETimes - News, 1998. http://www.eetimes.com/news/98/1000news/digital.html.

33. K.H. Tsoi, K.H. Leung and P.H.W. Leong., *Compact FPGA-based True and Pseudo Random Number Generators.,* Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, NT Hong Kong.

34. V. Fischer and M. Drutarovsky. *True random number generator embedded in reconfigurable hardware*. Trong Proceedings Cryptographic Hardware and Embedded Systems Workshop (CHES), trang 415-430, 2002.