Improved Algebraic Cryptanalysis of QUAD, Bivium and Trivium via Graph Partitioning on Equation Systems

Kenneth Koon-Ho Wong[1], Gregory V. Bard[2]

[1] Information Security Institute, Queensland University of Technology, Brisbane QLD 4000, Australia
kk.wong@qut.edu.au
[2] Mathematics Department, Fordham University, The Bronx NY 10458, USA
bard@fordham.edu

Abstract. We present a novel approach for solving systems of polynomial equations via graph partitioning. The concept of a variable-sharing graph of a system of polynomial equations is defined. If such graph is disconnected, then the corresponding system of equations can be split into smaller ones that can be solved individually. This can provide a significant speed-up in computing the solution to the system, but is unlikely to occur either randomly or in applications. However, by deleting a certain vertices on the graph, the variable-sharing graph could be disconnected in a balanced fashion, and in turn the system of polynomial equations are separated into smaller ones of similar sizes. In graph theory terms, this process is equivalent to finding balanced vertex partitions with minimum-weight vertex separators. The techniques of finding these vertex partitions are discussed, and experiments are performed to evaluate its practicality for general graphs and systems of polynomial equations. Applications of this approach in algebraic cryptanalysis on symmetric ciphers are presented. For the QUAD family of stream ciphers, we show how a malicious party can manufacture conforming systems that can be easily broken. For the stream cipher Trivium and its variants, we achieve significant speedups in algebraic attacks launched against them. In each of these cases, the systems of polynomial equations involved are well-suited to our graph partitioning method. These results may open a new avenue for evaluating the security of symmetric ciphers against algebraic attacks.

1 Introduction

There has been a long history of the use of graph theory in solving systems of equations. Graph partitioning techniques are applied to processes such as reordering variables in matrices to reduce fill-in for sparse systems [20, Ch. 7] and partitioning a finite element mesh across nodes in parallel computations [43]. These techniques primarily focus on linear systems over the real or complex fields. In this paper, we apply similar graph theory techniques to systems of multivariate polynomial equations, and develop methods of partitioning these systems into ones of smaller sizes via their "variable-sharing" graphs. These techniques are intended to work over any field, finite or infinite, but are particularly suited to GF(2) for use in algebraic cryptanalysis of symmetric ciphers. In most algebraic cryptanalysis, the symmetric ciphers are described by systems of polynomial equations over $GF(2)$. The graph theory methods introduced in this paper can be used to improve the efficiency of solving these systems of equations, which translate to a possible reduction of the security of these ciphers. This will be exemplified with the QUAD [11] and Trivium [21] stream ciphers.

Computing the solution to a system of multivariate polynomial equations is an NP-complete problem [7, Ch. 3.9]. A variety of solution techniques have been developed for solving these polynomial systems over finite fields, such as linearization, Gröbner bases, and resultants [8, Ch. 12], as well as recent ones such as SAT-solvers [9] and the Raddum-Semaev method [49]. Over the real and complex numbers, numerical techniques are also known including Gradient Descent, Newton's Method, the Conjugate Gradient Methods and the Nelder-Mead Simplices Algorithm [6]. In addition, homotopy methods, also known as continuation methods, have become popular [3], but require the field to be ordered and complete. The graph partitioning method introduced in this paper could be a novel addition to the variety of methods available, principally as a preprocessor.

From a multivariate polynomial system of equation, a variable-sharing graph is constructed with a vertex for each variable in the system, and an edge between two vertices if and only if those variables appear together in any equation in the system. Clearly, if the graph is disconnected, the system can be split into two separate systems of smaller sizes, and they can be solved for individually. However, even if the graph is connected, we show that it may be possible to disconnect the graph by eliminating a few variables by, for example, guessing their values when computing over a small finite field, and thereby splitting the remaining system. This suggests a divide-and-conquer approach to solving systems of equations. When the polynomial terms in the an system of equations are very sparse, we show that the system can usually be reduced to a set of smaller systems, whose solutions can be computed individually in much less time.

In order for a partition of a system to be productive, the minimum number of variables should be eliminated, and the two subsystems must be approximately equal in size. This ensures that the benefit of partitioning the system is maximised. These conditions lead to the problem of finding a balanced vertex partition with a minimum-weight vertex separator on its variable-sharing graph, which is an NP-complete problem [34, 44]. Nevertheless, heuristic algorithms can often find near-optimal partitions efficiently [36].

In this paper, we offer two cryptographic applications of vertex partitioning arising from algebraic cryptanalysis of stream ciphers, where both achieve positive results. First, we describe a method whereby a manufacturer of a sparse implementation of QUAD [11], a provably-secure stream cipher family, could "poison" the polynomial system in the cipher, and thereby enable messages transmitted with it to be read by the manufacturer. Second, we present an algebraic cryptanalysis of Trivium [21], a profiled stream cipher in the eSTREAM project, as well as its reduced versions Bivium and Bivium-A, and discuss the implications of graph partitioning methods of solving the corresponding systems of equations.

Section 2 introduces the necessary background in graph theory and graph partitioning. Section 3 shows how a system of polynomial equations can be split into ones of smaller sizes using graph partitioning methods. Section 4 provides results for some partitioning experiments and analyses the feasibility of equation solving via graph partitioning methods. Section 5 presents the applications of graph partitioning methods on the algebraic cryptanalysis of QUAD and Trivium. Conclusions will be drawn in Section 7. Appendix A discusses the NP-completeness of finding balanced graph partitions. Some theorems guaranteeing the existence of balanced graph partitions for special graphs are given in Appendix B. An algorithm to convert from edge partitions to vertex partitions is given in Appendix C.

2 Preliminaries

In this section, a brief introduction to graphs and graph partitioning is presented. For a detailed treatment on graph theory, see [31]. A graph describes a set of nodes and connections between them. Each node is called a vertex, and a connection between two nodes is called an edge.

Let $G = (V, E)$ be a graph with vertex set V and edge set E. Two vertices $v_i, v_j \in V$ are connected if there is a path from v_i to v_j through edges in E. A disconnected graph is a graph where there exists at least one pair of vertices that is not connected, or if the graph has only one vertex.

A graph $G_1 = (V_1, E_1)$ with vertex set $V_1 \subseteq V$ and edge set $E_1 \subseteq E$ is called a subgraph of G. Given a graph G, subgraphs of G can be obtained by removing vertices and edges from G. Let $G = (V, E)$ be a graph with k vertices and l edges, such that $V = \{v_1, v_2, \ldots, v_{k-1}, v_k\}$, $E = \{(v_{i_1}, v_{j_1}), (v_{i_2}, v_{j_2}), \ldots, (v_{i_l}, v_{j_l})\}$. Removing a vertex v_k from V forms a subgraph $G_1 = (V_1, E_1)$ with $V_1 = \{v_1, v_2, \ldots, v_{k-1}\}$ and $E_1 = \{(v_i, v_j) \in E \mid v_k \notin \{v_i, v_j\}\}$. We call G_1 the subgraph of G induced by the vertex set $(V - \{v_k\})$.

Let $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ be two subgraphs of G. G_1, G_2 are considered disjoint if no vertices in G_1 are connected to vertices in G_2. Clearly, the condition $V_1 \cap V_2 = \emptyset$ is necessary.

2.1 Graph Connectivity

The goal of partitioning a graph is to make the graph disconnected by removing some of its vertices or edges. The number of vertices or edges that needs to be removed to disconnect a graph is related to its vertex or edge connectivities respectively.

Definition 2.1. *The* vertex connectivity $\kappa(G)$ *of a graph G is the minimum number of vertices that must be removed to disconnect G. The* edge connectivity

$\lambda(G)$ of G is the minimum number of edges that must be removed to disconnect G.

Clearly, a disjoint graph has vertex connectivity zero. On the other extreme, a complete graph K_n, where all n vertices are connected to each other, has vertex connectivity $(n-1)$. The removal of all but one vertex from K_n results in a graph consisting of a single vertex, which is considered to be disconnected.

2.2 Graph Partitioning

The process of removing vertices or edges to disconnect a graph is called vertex partitioning or edge partitioning respectively. All non-empty graphs admit trivial vertex and edge partitions, where all connections to a single vertex are removed. This is obviously not useful for most applications. In this paper, we only consider balanced partitions with minimum-weight separators, in which a graph is separated into subgraphs of roughly equal sizes by removing as few vertices or edges as possible.

Balanced Vertex Partitions More specifically, our primary focus of this paper is on balanced vertex partitions.

Definition 2.2. *Let $G = (V, E)$ be a graph. A vertex partition (V_1, C, V_2) of G is a partition of V into mutually exclusive and collectively exhaustive sets of vertices V_1, C, V_2, where V_1, V_2 are non-empty, such that no edges connect vertices of V_1 directly to vertices of V_2. The removal of C causes the subgraphs induced by V_1 and V_2 to be disjoint, hence C is called the vertex separator.*

For a balanced vertex partition, we require V_1 and V_2 to be of similar size. For a minimum-weight separator, we also require that C be sufficiently small. This is to ensure that the vertex partition obtained is useful for its applications, otherwise much of the original information would be lost.

Definition 2.3. *Let $G = (V, E)$ be a graph, and (V_1, C, V_2) be a vertex partition of G with vertex separator C (see Definition 2.2). If $\max(|V_1|, |V_2|) \leq \alpha|V|$, then G is said to have an α-vertex separator.*

The problem of finding α-vertex separators is known to be NP-hard. For more details, see Appendix A.

Definition 2.4. *Let $G = (V, E)$ be a graph. If (V_1, C, V_2) is a vertex partition of G, then define*

$$\beta = \frac{\max(|V_1|, |V_2|)}{|V_1| + |V_2|} = \frac{\max(|V_1|, |V_2|)}{|V| - |C|}$$

to be a measure of balance of the vertex partition.

Suppose the balance of a vertex partition of G into (V_1, C, V_2) is $\beta = \alpha$, then the partition also satisfies $\max(|V_1|, |V_2|) = \alpha(|V_1| + |V_2|) \leq \alpha|V|$, and hence the G has an α-vertex separator. Therefore, theorems that apply to α-vertex separators would also apply to vertex partitions with balance α. See [44] for more details of α-vertex separators. Several theoerems governing the existence of α-vertex separators are presented in Appendix B.

Figure 1 presents examples of balanced and unbalanced partitions, and their respective β values. The vertex separators C are circled, with the partitioned vertices V_1, V_2 outside. The removal of the vertices in the separators disconnects the graphs. For a balanced partition, β should be close to 1/2.

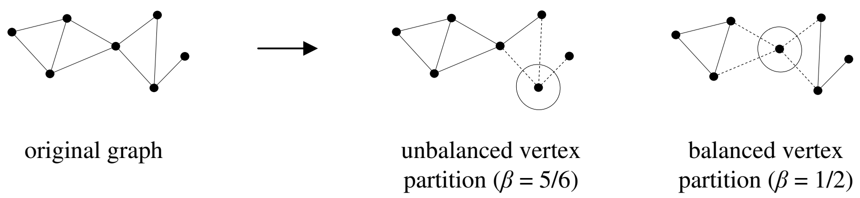

Fig. 1. Balanced and Unbalanced Vertex Partitions

Partitioning Algorithms and Software Balanced edge partitioning is widely used in scientific and engineering applications, such as electric circuit design [50], parallel matrix computations [39], and finite element analysis [43]. Software packages are readily available for computing balanced edge partitions using a variety of algorithms [10, 13, 27, 32, 45, 46, 53].

On the other hand, balanced vertex partitioning has fewer applications, one of which being variable reordering in linear systems [20]. We are not aware of publicly available software that could be used for directly computing balanced vertex partitions with minimum-weight vertex separators. Therefore, we have chosen to use an indirect method to compute vertex partitions from edge partitions obtained by software. More details will be discussed in Section 3.

Unless otherwise stated, from here on we will only consider the problem of balanced vertex partitioning with minimum-weight vertex separators (sometimes simply referred to as vertex partitioning or partitioning) and its applications to solving systems of multivariate polynomial equations.

3 Partitioning Polynomial Systems

In this section, our method for partitioning systems of multivariate polynomal equations by finding balanced vertex partitions of their variable-sharing graphs is described. Several methods for finding and using these partitions will also be discussed.

Definition 3.1. *Let F be the polynomial system*

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_m(x_1, x_2, \ldots, x_n) &= 0 \end{aligned}$$

of m polynomial equations in the variables $x_1, x_2, \ldots, x_n$. The variable-sharing *graph $G = (V, E)$ of F is obtained by creating a vertex $v_i \in V$ for each variable x_i, and creating an edge $(v_i, v_j) \in E$ if two variables x_i, x_j appear together in any polynomial f_k.*

Example 3.1. Suppose we have the following quadratic system of equations over GF(2), where the variables $x_1, x_2, \ldots, x_5$ are known to take values in GF(2).

$$\begin{aligned} x_1x_3 + x_1 + x_5 &= 1 \\ x_2x_4 + x_4x_5 &= 0 \\ x_1x_5 + x_3 &= 0 \\ x_2x_5 + x_2 + x_4 &= 0 \\ x_2 + x_4x_5 &= 1 \end{aligned} \tag{1}$$

The corresponding variable-sharing graph G and a balanced vertex partition is shown in Figure 2.

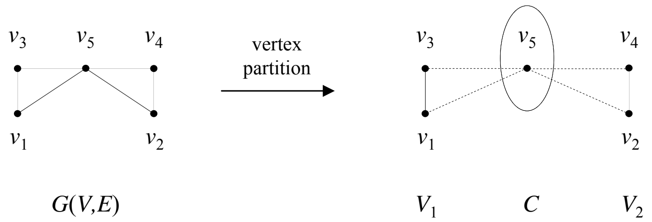

Fig. 2. Graph of quadratic system (1) and a vertex partition.

The quadratic system can then be partitioned into two systems of equations with the common variable x_5 as follows.

$$\begin{aligned} x_1x_3 + x_1 + x_5 &= 1 & x_2x_4 + x_4x_5 &= 0 \\ x_1x_5 + x_3 &= 0 & x_2x_5 + x_2 + x_4 &= 0 \\ & & x_2 + x_4x_5 &= 1 \end{aligned} \tag{2}$$

Since $x_5 \in \mathrm{GF}(2)$, we can substitute all possible values of x_5 into (1) and computing solutions to the reduced systems to give

$$\begin{aligned} x_5 = 0 &\Rightarrow \text{no solution} \\ x_5 = 1 &\Rightarrow x = (1, 0, 1, 0, 1) \end{aligned}$$

The solution obtained is the same as if we had directly computed the solution to the full system of equations. However, the systems have been reduced to having less than half the number of variables compared to the the original, at the cost of applying guesses to one variable.

This method of guessing and solving will be used for the algebraic cryptanalysis of the Trivium stream cipher in Section 5. For simplicity, from here on we might use the terms for variables and vertices interchangeably to denote the variables in the polynomial systems of equations and their corresponding vertices in the variable-sharing graphs, provided there are no ambiguities.

3.1 Partitioning Algorithms

While balanced partitioning is an NP-hard problem, a variety of heuristic algorithms have been found to be very efficient in finding near-optimal partitions. Software implementing these algorithms and their applications are as discussed in Section 2.2.

One of the efficient schemes for balanced graph partitioning is called multilevel partitioning, which we have selected to use for our experiments. Suppose a graph G_0 is to be partitioned. Firstly, G_0 "coarsened" progressively into simpler graphs $G_1, G_2, \ldots, G_r$ by contracting adjacent vertices. The process of choosing vertices for contraction is called matching. After reaching a graph G_r with the desired level of simplicity, a partitioning is performed. The result is then progressively refined back through the chain of graphs $G_{r-1}, G_{r-2}, \ldots, G_0$. At each refining step, a refinement to the partition can be performed. The output is then a partition of G_0. Details of multilevel partitioning can be found in [33, 36]. Examples of partitioning and refinement algorithms include the ones by Kerighan-Lin [37] and Fiduccia-Mattheyses [26].

As discussed in Section 2, it appears that implementations of vertex partitioning are not readily available. This is also true for multilevel partitioning algorithms. Therefore, we have chosen to use the multilevel edge partitioning software Metis [35] for our study. The Matlab interface Meshpart [30] to Metis is used to access the algorithms. The interface Meshpart also contains a routine to convert an edge partition found by Metis to a vertex partition, which completes our software requirements for finding balanced vertex partitions. These will be used for the experiments in Section 4 and for the algebraic cryptanalysis of Trivium in Section 5.2.

4 Graph Partitioning Experiments

To evaluate the practicality of partitioning large systems of equations, experiments have been performed on random graphs of different sizes resembling typical variable-sharing graphs. These experiments were run on a Pentium M 1.4 GHz CPU with 1 GB of RAM using the Meshpart [30] Matlab interface to the Metis [35] partitioning software.

Definition 4.1. *Let $G = (V, E)$ be a graph. The* degree $\deg(v)$ *of a vertex $v \in V$ is the number of edges $e \in E$ incident (connecting) to v.*

Definition 4.2. *The* density $\rho(G)$ *of a graph $G = (V, E)$ is the ratio of the number of edges $|E|$ in G to the maximum possible number $\frac{1}{2}|V|(|V| - 1)$ of edges in G.*

In each experiment, random graphs $G = (V, E)$ are generated, each with prescribed number of vertices $|V|$, number of edges $|E|$, and average degree d of its vertices. Their densities ρ are also computed. For each graph, a vertex partition is performed to give (V_1, C, V_2), where C is the vertex separator. The balance measure β is then computed, and the time required is also noted. Some experimental results are shown in Table 1.

$\lvert V\rvert$	$\lvert E\rvert$	ρ	d	$\lvert C\rvert$	$\lvert V_1\rvert$	$\lvert V_2\rvert$	β	Time
64	64	0.0308	2	5	31	28	0.5254	61.26 ms
64	128	0.0615	4	15	30	19	0.6122	63.06 ms
64	256	0.1231	8	26	28	10	0.7368	80.95 ms
64	512	0.2462	16	32	3	29	0.9063	67.36 ms
128	128	0.0155	2	7	64	57	0.5289	64.80 ms
128	256	0.0310	4	28	60	40	0.6000	66.73 ms
128	512	0.0620	8	55	45	28	0.6164	63.27 ms
128	1024	0.1240	16	62	63	3	0.9545	83.51 ms
1024	1024	0.0020	2	51	508	465	0.5221	74.58 ms
1024	2048	0.0039	4	222	482	320	0.6010	90.05 ms
1024	4096	0.0078	8	418	355	251	0.5858	113.66 ms
1024	8192	0.0156	16	509	511	4	0.9922	168.55 ms
4096	4096	0.0005	2	183	2039	1874	0.5211	122.48 ms
4096	8192	0.0010	4	877	1903	1316	0.5912	175.20 ms
4096	16384	0.0020	8	1697	1539	860	0.6415	289.24 ms
4096	32768	0.0039	16	2037	2047	12	0.9942	548.75 ms

Table 1. Vertex Partitioning Experiments

It can be observed from Table 1 that the graph of β is likely to be correlated with the average degree d of the graphs. Small vertex separators can be obtained when the number of edges is a small factor of the number of vertices. At $d = 16$, the value of β is near its upper bound of 1, which means that those partitions are unlikely to be useful. Since the maximum number of edges for a graph of size n is $O(n^2)$, the edge density must be smaller with a larger graph for practical partitions. This is a reasonable assumption for polynomial systems, since certain sparse systems have only a small number of variables in each equation, regardless of the total number of variables in the system. This fact is true for the case of Trivium in Section 5.

It is also noted that the time required to compute vertex partitions are quite short for the graph sizes considered, and would be negligible compared to the time required to solve the partitioned systems.

4.1 On the Partition Balance β

Figure 3 shows the effect of varying the number of edges in a graph on the balance β of vertex partitions for graphs of different sizes, where $n = |V|$. Each value of the partition balance parameter β shown in the figure is the average value of 200 separate graph partitions.

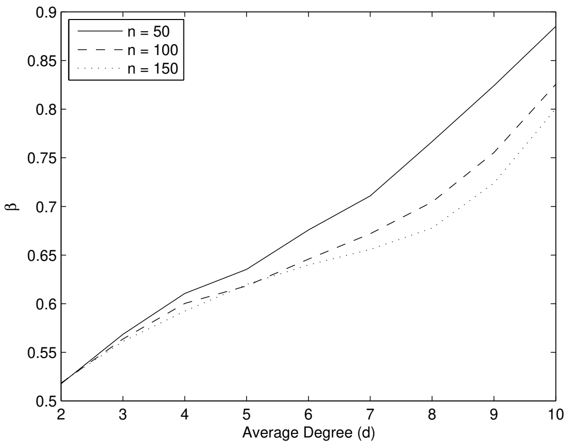

Fig. 3. Partition Balance vs Average Degree of Graph

It can be observed that the balance parameter β increases approximately linearly with the average degree d of the graphs. Furthermore, the increase seems to be slower with graphs having more vertices. However, Table 1 suggests that the increase becomes faster again for even larger graphs. From the vertex partition theorems shown in Appendix B, we assert that a reasonable balance is obtained when $\frac{1}{2} \leq \beta \leq \frac{2}{3}$. With the graph sizes considered in Figure 3, it seems that this reasonable balance can be obtained with $d \leq 6$. This also means that when $d \leq 6$, the graph may be planar, since planar graphs has $\frac{2}{3}$-vertex separators, which implies $\frac{1}{2} \leq \beta \leq \frac{2}{3}$. Marginally balanced partitions with $\beta < \frac{3}{4}$ are possible with $d \leq 9$.

5 Applications to Algebraic Cryptanalysis

A bit-based stream cipher usually consists of an internal state $s \in \mathrm{GF}(2)^n$ and a defined procedure to update s at each timestep t of the cipher. At the start of an encryption, a key initialisation phase would take place, whereby a secret key k and a known initialisation vector IV are used to set its s to its secret initial state s_0. The cipher then begins its keystream generation phase, and outputs a series of keystream bits $z_1, z_2, \ldots$ from s at each timestep t, where s is updated based on the defined procedure. This procedure can usually be described by $s_{t+1} = g(s_t)$ at each timestep t. Simliarly, the keystream bits z_t can usually be described by $z_t = f(s_t)$ at each timestep t. Given plaintext bits $p_1, p_2, \ldots$, the stream cipher encrypts them using the keystream into ciphertext bits $c_1, c_2, \ldots$ as $c_t = p_t + z_t$ over GF(2).

Since stream ciphers are traditionally designed for implementation in digital circuits, f, g can often be represented as polynomial functions. Then, if both p_t and c_t are known for enough timesteps, one can write a system of equations based on $z_t = p_t + c_t$ using f, g. This forms the foundation of algebraic cryptanalysis. To perform an algebraic cryptanalysis of a stream cipher, the cipher is first described as a system of equations. Its variables usually correspond to the bits in the key k or the initial state s_0. If the variables are from k, solving the system is called "key recovery", and the cipher is immediately broken. If the variables are from s_0, solving the system is called "state recovery", and the key could be derived from the solution, whose difficulty depends on the specific cipher design.

Every attack on every cipher has its nuances, and so above description is necessarily vague. For an overview of algebraic cryptanalysis, see [8]. For techniques of algebraic cryptanalysis on specific types of ciphers, see [19, 18, 16, 2, 55]. Some uses of graph theory for algebraic attacks can also be found in [48, 54]. In this section, two applications of our equation partitioning to algebraic cryptanalysis are presented. Firstly, we show a malicious use of the stream cipher QUAD [11]. Then, we describe and perform an algebraic cryptanalysis to the stream cipher Trivium [21] and its variants Bivium-A and Bivium-B [48]. We discuss only the equations arising from the cipher, and refer the reader to the respective references of these ciphers for their design and implementation details.

5.1 QUAD

The stream-cipher family QUAD is given in [11]. The security of QUAD is based on the Multivariate Quadratic (MQ) problem. The heart of the cipher is a random system of kn quadratic equations in n variables over a finite field GF(q). Usually, we have $q = 2$, but implementations with $q = 2^s$ have also been discussed [56]. This system of equations is not secret, but publicly known, and there are criteria for these equations, such as those relating to rank, which we omit here. In a different context, QUAD has been analyzed in [56, 5], and [8, Ch. 5.2].

Equations of QUAD The authors of QUAD recommend $k = 2$ and $n \geq 160$, so it is assumed that we have a randomly generated system of $2n = 320$ equations in $n = 160$ unknowns. The system is to be drawn uniformly from all those possible, which is to say that the coefficients can be thought of as generated by fair coins.

Each quadratic equation is a map $\mathrm{GF}(2)^n \to \mathrm{GF}(2)$, so the first set of n equations form a map $\mathrm{GF}(2)^n \to \mathrm{GF}(2)^n$ called f_1, and the second set of n equations also form a map of the same dimensions called f_2. The internal state is a vector s of 160 bits. The first 160 equations are evaluated at s, and the resulting vector $f_1(s_t) = s_{t+1}$ becomes the new state. The second 160 equations are evaluated to become the output of that timestep $z_t = f_2(s_t)$. The vector z_t is added to the next n bits of the plaintext p_t over GF(2), and is transmitted as the ciphertext $c_t = p_t + z_t$. There is also an elaborate setup stage which maps the secret key and an initialization vector to the initial state s_0.

Finding a pre-image under the maps f_1, f_2 i.e. finding s_i given s_{i+1} and z_i, is equivalent to solving a quadratic system of $2n$ equations in n unknowns, and is NP-hard [7, Ch. 3.9]. This is further complicated by the fact that the adversary would not have s_{i+1}, but rather only $z_i + p_i$.

Given a known-plaintext scenario, where the attacker knows both the plaintext $p_1, p_2 \ldots, p_n$ and ciphertext $c_1, c_2, \ldots, c_n$, one can write the following system of equations.

$$\begin{aligned}
c_1 + p_1 &= z_1 = f_2(s_1) \\
c_2 + p_2 &= z_2 = f_2(s_2) = f_2(f_1(s_1)) \\
c_3 + p_3 &= z_3 = f_2(s_3) = f_2(f_1(f_1(s_1))) \\
\vdots &= \vdots \\
c_t + p_t &= z_t = f_2(s_t) = f_2(f_1(\underbrace{f_1(f_1(\cdots f_1(}_{i \text{ times}} s_1) \cdots))))
\end{aligned}$$

The interesting fact here is that $f_2(f_1(f_1(\cdots f_1(s_1)\cdots))))$ and higher iterates might be quite dense even if f_1 is sparse. The authors of QUAD have excellent security arguments when the polynomial system is generated by fair coins. However, it will have on average 6440.5 monomials per equation or roughly 2 million in the system, which would require a large gate count or would be slow in software. Thus, in their conference presentation but not the paper, the authors of QUAD mention that a slightly sparse f might still be secure. Furthermore, because of repeated iteration and the general difficulty of the MQ problem, there would probably be no feasible algebraic attack against the sparse version.

Poisoned Equations and QUAD One could imagine the following scenario, which is inspired by Jacques Patarin's system "Oil and Vinegar" [38]. A malicious manufacturer does not generate the system at random, but rather creates a system that is sparse and has vertex connectivity of 20, for some vertex partition with $\beta \approx 0.6$. Our experiments in Section 4 show that this is a feasible partition. The malicious manufacturer would claim that the system is sparse for efficiency reasons and it might have a considerably faster encryption throughput than a QUAD system with quadratic equations generated by fair coins.

Some separators of 20 vertices divides the variable sharing graph into roughly 56 and 84 vertices. This means that an attacker would need only to know the plaintext and ciphertext of one 160-bit sequence, and solve the equation

$$f_2(\underbrace{f_1(f_1(\cdots f_1(f_1(}_{i-1 \text{ times}} s_1))\cdots))) = p_t + c_t$$

.

For any guess of the key, this would be solving 56 equations in 56 unknowns and 84 equations in 84 unknowns. Such a problem is certainly trivial for a SAT-solver, as shown in [9], [7, Ch. 3] and [8, Ch. 7]. Only 2^{20} iterations would be required, and with a massive parallel network, such as BOINC [1], this would be quite feasible.

Remedy to Poisoned Systems for QUAD While finding a balanced vertex partition of a graph G is NP-hard, as discussed in Appendix A, calculating the vertex connectivity $\kappa(G)$ is easier. If $\kappa(G) > 80$, for example, then there is no vertex partition, balanced or otherwise, with fewer than 80 vertices in the vertex separator. Then, by calculating $\kappa(G)$, a manufacturer of QUAD could prove that they are not poisoning the quadratic system as explained in the previous subsection. There are also techniques to generate functions with verifiable randomness [15], which could be used to construct polynomial systems of equations for QUAD, such that they are provably not poisoned.

5.2 Trivium

Trivium [21] is a bit-based stream cipher in the eSTREAM project portfolio for hardware implementation with an 80-bit key, 80-bit initialization vector, and a 288-bit internal state. As at the end of the eSTREAM project, after three phases of expert and community reviews, no feasible attacks faster than an exhaustive key search on the full implementation of Trivium were found. However, Trivium without key initialisation, as well as its reduced versions Bivium-A and Bivium-B with a 177-bit internal state, admit attacks faster than exhaustive key search. Cryptanalytic results on Trivium and Bivium have been presented in [12, 22, 23, 24, 41, 42, 47, 51].

Equation Construction The equations governing keystream generation from the initial state s_0 can be found in [21] for Trivium and [48] for Bivium. In the algebraic cryptanalysis presented in this paper, we do not consider the initialisation phase from the key k and initialisation vector IV, and hence we are performing state recovery of the cipher.

Trivium can be described as a system of 288 multivariate polynomial equations in 288 variables, but we found that this is too dense for partitioning to be useful. Instead, we use the quadratic equations presented in [48], which contains more variables, but are very sparse. The quadratic system of Trivium consists of 954 sparse quadratic equations in 954 variables, and observed keystream from 288 clocks. Similarly, the polynomial system of Bivium-A and Bivium-B consists of 399 sparse quadratic equations in 399 variables, and observed keystream from 177 clocks. There are at most 6 variables present in each equation, hence the variable-sharing graph has maximum degree 6, and there is at most one quadratic term in an equation. We attempt to solve these equations via partitioning.

Equation Partitioning The sparse quadratic equations for Trivium and Bivium are constructed as per [48], and their variable-sharing graphs are then computed. Figure 4 shows the adjacency matrix for the variable-sharing graph of Trivium. The sparsity of this matrix appears promising for a reasonable partition. Graphs for Bivium are of similar sparsity.

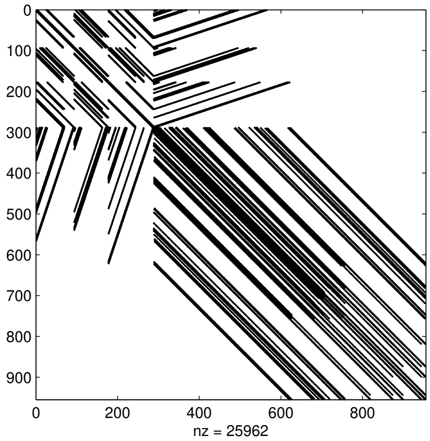

Fig. 4. Graph Adjacency Matrix of Trivium Equations

Partitioning these variable-sharing graphs $G = (V, E)$ into vertex sets V_1, V_2 and vertex separator C with [35] as in Section 4 gives the results shown in Table 2. From these results, it seems that both of the Bivium ciphers admit very

Cipher	State Size	Number of Variables	$\|C\|$	$\|V_1\|$	$\|V_2\|$	β
Bivium-A	177	399	96	156	147	0.5149
Bivium-B	177	399	122	150	127	0.5415
Trivium	288	954	288	476	190	0.7147

Table 2. Partitioning Equations of Bivium-A, Bivium-B and Trivium

balanced partitions, whereas Trivium did not. However, using an implementation of the greedy algorithm in Appendix C, we were able to find a balanced partition for Trivium with $|C| = 295$ and $\beta \approx 0.5$. This result is still preliminary, so we omit further details here.

The sizes of the vertex separators C are the number of variables that must be eliminated to separate the systems into two. In algebraic attacks, this corresponds to the number of variables whose values are to be discovered or guessed at a complexity of $2^{|C|}$. The process of guessing certain bits in order to find a solution is called partial key guessing. If the guessed bits are correct, then solving the remaining system would lead to the solution.

For Trivium, the separator size is exactly the same as the internal state size. For Bivium and Bivium-A, the separator sizes are less than the internal state size, but larger than the key size of 80-bits. This means that the time complexity of partial key guessing on all bits of the separators would be higher than that of a brute force search on the key.

Partial Key Guessing However, we can attempt to guess less bits then the size of the separator C. The remaining system would not be separated, but it can still be solved. We have found by experiment that a partial key guess on a subset of bits in C provides a significant advantage over that on random bits, in that the reduced polynomials systems are much faster to solve. These experiments were performed using Magma 2.12 [14] with its implementation of the Gröbner basis algorithm F_4 [25] for solving the reduced polynomial systems. The results are shown in Table 3, where n is the number of bits guessed, m is the number of equations resulting from the guess, with q of them being quadratic. Correct guesses are always used to reduce the polynomial systems, which means that the time and memory use presented are for solving the entire system arriving at a unique solution. All values are averaged over at most 10 individual runs.

The experimental results show that the time required for partial key guessing on n bits is reduced significantly if those bits are taken from the separator. This means that, by finding partitions to the system of equations, we have reduced the resistance of these ciphers to algebraic cryptanalysis, since a feasible partial key guess attack can potentially be launched on less bits with this extra information. For example, with Bivium-B, the time to compute a solution by guessing 78 bits randomly is roughly equivalent to that by guessing 66 bits in the separator. Hence, the time complexity for an attack on Bivium-B is reduced from $2^{78}T_B$ to $2^{66}T_B$ with the use of the separator, where T_B denotes the time complexity required to compute a solution to a reduced system of Bivium-B. For Trivium, the improvement is even more pronounced. The time complexity could be reduced from $2^{280}T_C$ to $2^{178}T_C$, which T_C denotes the time complexity required to compute a solution to a reduced system of Trivium.

In an actual algebraic attack, many of the guesses will result in inconsistent equations with no solutions, which can be checked and discarded easily. This means that the time required to process a guess is at most T. A full attack attempt was launched on Bivium-A with a partial key guess on 20 bits in its separator. About 200000 guesses of out the possible 2^{20} were made, with each guess taking on average about 0.15 seconds to process. This is much faster than the 45 seconds required from the experimental results to process a correct guess.

A Bit-Leakage Attack There is another scenario whereby the graph partitioning would provide an advantage to algebraic cryptanalysis. Suppose by some means, accidental or deliberate, some bits of the internal state could be leaked to an attacker. This would occur in a side-channel attack setting. If the attacker could control which bits are leaked, then the best choices would be those variables in the separator. Fewer bits would need to be leaked before the system of equations can be solved in a reasonable time.

6 Discussion

It can be observed that the stream ciphers QUAD and Trivium are susceptible to the graph partitioning method due to the sparsity of the equations systems describing these ciphers. This in turn means that the vertex connectivity of these systems are not optimal. As mentioned before, a complete graph has maximum vertex connectivity, and its corresponding equation system would be immune from this partitioning method. Therefore, for maximum protection against an algebraic attack of this kind, a cipher should be designed such that the all variables appear at least once with all other possible variables in the equation system, so that its corresponding graph is complete and does not admit a useful balanced vertex partition. In the case of Trivium and its variants, since the variable relabelling technique is used (see [21]) to generate the quadratic equations describing them, not all variables can appear together, and so the system is necessarily sparse. Further investigation is needed to examine the design criteria for such ciphers again this partitioning technique.

7 Conclusions

In this paper, the concept of a variable-sharing graph of a system of polynomial equations was defined. It has been shown that this concept can be used to break systems of polynomial equations into useful pieces, which can be solved for separately, provided that the graph has a vertex partition satisfying various requirements, namely that the vertex separator should be small, and the partition should be balanced. We also presented methods for finding the partition, and methods for using the partition to solve polynomial systems of equations over small and large fields more efficiently.

It has been shown that balanced vertex partitions are feasible to obtain for sparse systems of polynomial equations. Experiments on random graphs of reasonable size and sparsity resembling variable-sharing graphs of equation systems have been performed.

The practicality of this partitioning technique has been demonstrated in the algebraic cryptanalysis of the stream cipher Trivium and its reduced versions, where we have found balanced partitions of useful sizes. These partitions provide information for launching more effective algebraic attacks with partial key guessing, and improves the attack time by at least a few orders of magnitude. Furthermore, we show how the partitioning technique can be used to poison the provably secure stream cipher QUAD, so that a malicious manufacturer can recover the keystream much more efficiently.

As discussed earlier, this paper has provided a novel technique for preprocessing large sparse systems of equations, which could be used together with popular techniques such as Gröbner basis methods to significantly reduce the time for computing solutions to these systems. It has also been shown that this technique provides improvements to algebraic cryptanalysis, and further research into this area is warranted, since there may be security implications for further ciphers that are susceptible to this technique.

References

[1] BOINC: Berkeley Open Infrastructure for Network Computing. `http://boinc.berkeley.edu/`.

[2] S. Al-Hinai, L. Batten, B. Colbert, and K. K.-H. Wong. Algebraic attacks on clock-controlled stream ciphers. In L. M. Batten and R. Safavi-Naini, editors, *11th Australasian Conference on Information Security and Privacy — ACISP 2006*, volume 4058 of *Lecture Notes in Computer Science*, pages 1–16, Melbourne, Australia, 2006. Springer.

[3] E. L. Allgower and K. Georg. *Introduction to Numerical Continuation Methods*, volume 45 of *Classics in Applied Mathematics*. Society for Industrial Mathematics, 1987.

[4] N. Alon, P. Semour, and R. Thomas. A separator theorem for graphs with an excluded minor and its applications. *Journal of the American Mathematical Society*, 3(4):801–808, Oct. 1990.

[5] D. Arditti, C. Berbain, O. Billet, H. Gilbert, and J. Patarin. QUAD: Overview and recent developments. In E. Biham, H. Handschuh, S. Lucks, and V. Rijmen, editors, *Symmetric Cryptography*, volume 07021 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[6] M. Avriel. *Nonlinear Programming: Analysis and Methods*. Dover, 2003.

[7] G. V. Bard. *Algorithms for solving linear and polynomial systems of equations over finite fields with applications to cryptanalysis*. PhD thesis, Department of Applied Mathematics and Scientific Computation, University of Maryland at College Park, Aug. 2007. Available at `http://www.math.umd.edu/~bardg/bard_thesis.pdf`.

[8] G. V. Bard. *Algebraic Cryptanalysis*. Springer, 2009.

[9] G. V. Bard, N. Courtois, and C. Jefferson. Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF(2) via SAT-Solvers. Cryptology ePrint Archive, Report 2007/024, 2007. `http://eprint.iacr.org/2007/024.pdf`.

[10] R. Baños, C. Gil, J. Ortega, and F. G. Montoya. Multilevel heuristic algorithm for graph partitioning. In *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*. Springer, 2003.

[11] C. Berbain, H. Gilbert, and J. Patarin. QUAD: A practical stream cipher with provable security. In S. Vaudenay, editor, *Advances in Cryptology - Eurocrypt 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 109–128. Springer, 2006.

[12] D. Bernstein. Response to slid pairs in Salsa20 and Trivium. Technical report, 2008. `http://cr.yp.to/snuffle/reslid-20080925.pdf`.

[13] J. Berry, N. Dean, M. Goldberg, G. Shannon, and S. Skiena. Graph computation with LINK. *Software: Practice and Experience*, 30:12851302, 2000.

[14] W. Bosma, J. Cannon, and C. Playoust. The MAGMA algebra system. I. The user language. *Journal of Symbolic Computation*, 24(3-4):235–265, 1997.

[15] M. Chase and A. Lysyanskaya. Simulatable vrfs with applications to multi-theorem nizk. In *Advances in Cryptology - CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 303–322. Springer, 2007.

[16] J. Y. Cho and J. Pieprzyk. Algebraic attacks on SOBER-t32 and SOBER-t16 without stuttering. In B. Roy and W. Meier, editors, *Fast Software Encryption*, volume 3017 of *Lecture Notes in Computer Science*, pages 49–64, Delhi, India, 2004. Springer.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[18] N. Courtois. Algebraic attacks on combiners with memory and several outputs. In C. Park and S. Chee, editors, *Information Security and Cryptology - ICISC 2004*, volume 3506 of *Lecture Notes in Computer Science*, Seoul, Korea, 2004. Springer.

[19] N. Courtois and W. Meier. Algebraic attacks on stream cipher with linear feedback. In E. Biham, editor, *Advances in Cryptology - Eurocrypt 2003*, volume 2656, Warsaw, Poland, 2003. Springer.

[20] T. A. Davis. *Direct methods for sparse linear systems*, volume 2 of *Fundamentals of Algorithms*. SIAM, Philadelphia, USA, 2006.

[21] C. De Cannière and B. Preneel. Trivium specifications. Technical report, Katholieke Universiteit Leuven, 2007. `http://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf`.

[22] I. Dinur and A. Shamir. Cube attacks on tweakable black box polynomials. In *Advances in Cryptology - Eurocrypt 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 278–299. Springer, 2009.

[23] N. Eén and N. Sörensson. Minisat — a SAT solver with conflict-clause minimization. In F. Bacchus and T. Walsh, editors, *Proc. Theory and Applications of Satisfiability Testing (SAT'05)*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer-Verlag, 2005.

[24] T. Eibach, E. Pilz, and G. Völkel. Attacking Bivium using SAT solvers. In H. K. Büning and X. Zhao, editors, *Theory and Applications of Satisfiability Testing (SAT '08)*, volume 4996 of *Lecture Notes in Computer Science*, pages 63–76. Springer-Verlag, 2008.

[25] J.-C. Faugère. A new efficient algorithm for computer Gröbner bases (f_4). *Journal of Pure and Applied Algebra*, 139:61–88, 1999.

[26] C. Fiduccia and R. Mattheyses. A linear time heuristic for improving network partitions. In *19th ACM/IEEE Design Automation Conference*, pages 175–181, 1982.

[27] C. Fremuth-Paeger. Goblin: A graph object library for network programming problems, 2007. http://goblin2.sourceforge.net/.

[28] M. R. Garey, P. S. Johnson, and L. Stockmeyer. Simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[29] J. R. Gilbert, J. P. Hutchinson, and R. E. Tarjan. A separation theorem for graphs of bounded genus. *Journal of Algorithms*, 5:391–407, 1984.

[30] J. R. Gilbert and S.-H. Teng. Meshpart: Matlab mesh partitioning and graph separator toolbox, 2002. `http://www.cerfacs.fr/algor/Softs/MESHPART`.

[31] J. L. Gross and J. Yellen, editors. *Handbook of Graph Theory*, volume 25 of *Discrete Mathematics and its Applications*. CRC Press, New York, USA, 2003.

[32] B. Hendrickson and R. Leland. The Chaco user's guide: Version 2.0. Technical Report SAND94-2692, Sandia National Laboratories, 1994.

[33] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *1995 ACM/IEEE Supercomputing Conference*. ACM, 1995.

[34] D. S. Johnson. The NP-completeness column: An on-going guide. *J. Algorithms*, 8:438–448, 1987.

[35] G. Karypis et al. Metis — Serial graph partitioning and fill-reducing matrix ordering, 1998. `http://glaros.dtc.umn.edu/gkhome/views/metis/`.

[36] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999.

[37] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphics. *Bell Systems Technical Journal*, 49:291–307, 1970.

[38] A. Kipnis, J. Patarin, and L. Goubin. Unbalanced oil and vinegar signature schemes. In *EUROCRYPT*, pages 206–222, 1999.

[39] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin/Cummings Publishing Company, Redwood City, CA, 1994.

[40] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, Apr. 1979.

[41] A. Maximov and A. Biryukov. Two trivial attacks on Trivium. In C. M. Adams, A. Miri, and M. J. Wiener, editors, *Proc. Selected Areas in Cryptography (SAC07)*, volume 4876 of *Lecture Notes in Computer Science*, pages 36–55. Springer-Verlag, 2007. Available from `http://eprint.iacr.org/2007/021`.

[42] C. McDonald, C. Charnes, and J. Pieprzyk. An algebraic analysis of Trivium ciphers based on the boolean satisfiability problem. Cryptology ePrint Archive, Report 2007/129, 2007. `http://eprint.iacr.org/2007/129`. Presented at the International Conference on Boolean Functions: Cryptography and Applications (BFCA2008).

[43] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Automatic mesh partitioning. In A. George, J. Gilbert, , and J. Liu, editors, *Graph Theory and Sparse Matrix Computation*, volume 56 of *The IMA Volumes in Mathematics and its Application*, pages 57–84. Springer, 1993.

[44] R. Müller and D. Wagner. α-vertex separator is NP-hard even for 3-regular graphs. *J. Computing*, 46:343–353, 1991.

[45] F. Pellegrini and J. Roman. SCOTCH: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *HPCN'96*, volume 1067 of *LNCS*, pages 493–498, Brussels, Belgium, 1996. Springer.

[46] R. Preis and R. Diekmann. The PARTY partitioning-library, user guide - version 1.1. Technical Report tr-rsfb-96-024, University of Paderborn, 1996.

[47] D. Priemuth-Schmid and A. Biryukov. Slid pairs in Salsa20 and Trivium. In D. R. Chowdhury, V. Rijmen, and A. Das, editors, *Progress in Cryptology—INDOCRYPT'08*, volume 5365 of *Lecture Notes in Computer Science*, pages 1–14. Springer-Verlag, 2008.

[48] H. Raddum. Cryptanalytic results on Trivium. Technical Report 2006/039, The eSTREAM Project, 27 March 2006. `http://www.ecrypt.eu.org/stream/papersdir/2006/039.ps`.

[49] H. Raddum and I. Semaev. New technique for solving sparse equation systems. Cryptology ePrint Archive, Report 2006/475, 2006. `http://eprint.iacr.org/2006/475`.

[50] D. G. Schweikert and B. W. Kernighan. A proper model for the partitioning of electrical circuits. In *9th workshop on Design automation*, pages 57–92. ACM, 1972.

[51] M. Vielhaber. Breaking One.Fivium by AIDA an algebraic IV differential attack. Cryptology ePrint Archive, Report 2007/413, 2007. `http://eprint.iacr.org/2007/413`.
[52] D. Wagner and F. Wagner. Between min-cut and graph bisection. Technical Report B-91-1, Freie Universität Berlin, 1991.
[53] C. Walshaw and M. Cross. JOSTLE: Parallel Multilevel Graph-Partitioning Software - An Overview. Technical report, Civil-Comp Ltd., 2007.
[54] K. K.-H. Wong. *Application of Finite Field Computation to Cryptology: Extension Field Arithmetic in Public Key Systems and Algebraic Attacks on Stream Ciphers.* PhD Thesis, Information Security Institute, Queensland University of Technology, 2008.
[55] K. K.-H. Wong, B. Colbert, L. Batten, and S. Al-Hinai. Algebraic attacks on clock-controlled cascade ciphers. In R. Barua and T. Lange, editors, *Progress in Cryptology - Indocrypt 2006*, volume 4329, pages 32–47, Kolkata, India, 2006. Springer.
[56] B.-Y. Yang, O. C.-H. Chen, D. J. Bernstein, and J.-M. Chen. Analysis of QUAD. In *Fast Software Encryption*, volume 4593 of *Lecture Notes in Computer Science*, pages 290–308. Springer, 2007.

A NP-Completeness of the Problem

Both the problems of finding balanced vertex partitions and balanced edge partitions are known to be NP-complete. For balanced vertex partitions, these are equivalent to the α-vertex separator decision and optimization problems, and were proven NP-Complete/NP-hard respectively by Müller and Wagner in [44]. For $\alpha = \frac{1}{2}$, the problems were proven NP-complete/NP-hard by Johnson in [34], by reduction to the known NP-complete problem "Balanced Complete Bipartite Subgraph Problem". In the case of edge partitions, these become the α-edge separator decision and optimization problems, which can be defined similarly. The were proven NP-Complete/NP-hard respectively by Wagner and Wagner in [52]. The $\alpha = \frac{1}{2}$ case was shown to be NP-Complete/NP-hard by Garey Johnson and Stockmeyer, in [28], as the "Minimum-Bisection Problem.".

B Existence Theorems for Vertex Partitions

Several theorems govern the existence of balanced vertex partitions and small vertex cuts for specific classes of graphs. These include planar graphs, graphs of a certain genus and graphs with specific structures. Throughout this section, let $G = (V, E)$ be a graph, and (V_1, C, V_2) be a vertex partition of G with vertex separator C.

Definition B.1. *A graph G is* planar *if it can be embedded in a plane without graph edges crossing, i.e. it can be drawn in a plane without any edge crossing another.*

Theorem B.1 (Lipton and Tarjan [40], 1979). *If G is a planar graph, there is a vertex partition with $|C| \leq \sqrt{8|V|}$ such that $|V_1| \leq \frac{2}{3}|V|$ and $|V_2| \leq \frac{2}{3}|V|$.*

This implies G has a $\frac{2}{3}$-vertex separator. A graph that cannot be drawn without edges crossing on a plane, but can be so drawn on a torus is said to be genus 1. This can be generalized as follows.

Definition B.2. *A graph G is said to be* genus g *if it can be drawn without edges crossing on a surface of topological genus g but not on any surface of smaller topological genus.*

Theorem B.2 (Gilbert, Hutchinson and Tarjan [29], 1984). *If G is a graph of genus $g > 1$, there is a vertex partition with $|C| = O(\sqrt{g|V|})$.*

The following theorem relates edge contraction and graph minors with vertex cuts. Contracting an edge between two vertices v_i, v_j means creating a new vertex v_k, such that any edge to either v_i or v_j now goes to v_k instead, and then both v_i, v_j are deleted. A graph G is said to have a minor K if some subgraph of G is isomorphic to K after contracting zero or more edges.

Theorem B.3 (Alon, Seymour and Thomas [4], 1990). *If a graph has no K_h minor, then there is a cut with $|C| \leq \sqrt{h^3|V|}$ such that*[3] *$|V_1| \leq \frac{2}{3}|V|$ and $|V_2| \leq \frac{2}{3}|V|$.*

Again, this implies G has a $\frac{2}{3}$-vertex separator. Determining the largest h for a general graph is NP-hard, because it is related to the known NP-complete problem "Max-Clique" [17, Ch. 34].

In the special case of bounded-degree graphs, where each vertex has at most degree d_{max}, this last theorem is particularly useful to us. Since K_h has degree h at every vertex, a bounded-degree graph cannot have K_h as a subgraph if $h > d_{max}$. However, it may have K_h as a minor, as merging adjacent vertices can increase degree. Nonetheless, the experiments in Section 4 also show that low-degree graphs tend to have balanced vertex cuts, while high-degree random graphs do not.

Given that C is small, the conditions $|V_1| \leq |V_2| \leq \frac{2}{3}|V|$ would represent good balance. However, in most applications, the graphs that arise are usually large and have less favorable structures then the above (i.e. they have K_h minors for large values of h). Therefore, we rely on heuristics to compute these vertex partitions.

C From Edge Partition to Vertex Partition

Given an edge partition of a graph G with edge separator B, there may be many sets of vertices of various cardinalities which will give a similar vertex partition. In particular, this will happen if there exist at least one vertex that is incident to several edges in the partition.

Figure 5 shows a greedy algorithm for finding a vertex partition with small vertex separator C that is equivalent to a given edge partition with edge separator B. One starts with a set of edges D representing the edge separator. Then,

[3] Furthermore, it is conjectured in the same paper that the h^3 can become h^2 instead.

at each iteration, choose the vertex which is incident on the largest number of edges in D. Mark that vertex as "to be deleted", and then delete the edges in D that are incident upon that vertex. This process is repeated until D is empty.

Input: B, an edge separator of a graph $G = (V, E)$.
Output: C, a small vertex separator of G based on B.

1. $D \leftarrow B$.
2. $R \leftarrow \emptyset$.
3. While $D \neq \emptyset$ do:
   (a) Pick the vertex in V that is incident on the highest number of edges in D. Call it v.
   (b) Insert v into C.
   (c) Remove from D any edge that v is incident upon.
4. Return C.

Fig. 5. The Greedy Algorithm Approach to Converting a Balanced Edge Partition to a Balanced Vertex Partition

Clearly, this algorithm will produce disconnected components that are subsets of the original edge partition, but there may possibly be smaller unrelated vertex subsets which could have accomplished the same with fewer vertices.

Cipher	All Guesses in $\|C\|$	n	m	q	Time	Memory
Bivium-A	No	24	422	193	26 s	42 MB
Bivium-A	No	22	419	200	120 s	175 MB
Bivium-A	No	20	421	200	195 s	234 MB
Bivium-A	No	18	417	203	2558 s	843 MB
Bivium-A	Yes	24	422	187	1 s	22 MB
Bivium-A	Yes	22	420	190	1 s	22 MB
Bivium-A	Yes	20	419	193	45 s	89 MB
Bivium-A	Yes	18	417	195	80 s	127 MB
Bivium-A	Yes	16	415	201	1101 s	751 MB
Bivium-A	Yes	14	413	202	2023 s	1200 MB
Bivium-B	No	82	481	140	180 s	1044 MB
Bivium-B	No	80	479	143	392 s	1044 MB
Bivium-B	No	78	477	146	740 s	1044 MB
Bivium-B	No	76	475	141	1213 s	1044 MB
Bivium-B	Yes	74	473	128	4 s	35 MB
Bivium-B	Yes	70	469	132	12 s	62 MB
Bivium-B	Yes	66	465	136	623 s	546 MB
Bivium-B	Yes	62	461	141	3066 s	1569 MB
Trivium	No	280	1333	329	13 s	80 MB
Trivium	No	276	1228	341	110 s	308 MB
Trivium	No	272	1224	343	155 s	554 MB
Trivium	No	268	1221	344	125 s	576 MB
Trivium	No	264	1217	344	594 s	1569 MB
Trivium	No	260	1213	344	747 s	3600 MB
Trivium	Yes	190	1140	493	14 s	584 MB
Trivium	Yes	184	1135	497	16 s	596 MB
Trivium	Yes	180	1131	499	18 s	596 MB
Trivium	Yes	178	1130	499	18 s	596 MB
Trivium	Yes	176	1127	499	4511 s	1875 MB
Trivium	Yes	174	1126	501	10543 s	3150 MB

Table 3. Partial Key Guessing on Trivium and Bivium